

---

# **ST10X167**

## **USER'S MANUAL**

Release 1.1



---

---

<b>TABLE OF CONTENTS</b>		<b>Page</b>
<b>1 -</b>	<b>INTRODUCTION .....</b>	<b>10</b>
1.1 -	DIFFERENCES BETWEEN THE ST10R167, ST10C167 & ST10F167 .....	11
1.2 -	ABBREVIATIONS.....	12
<b>2 -</b>	<b>ARCHITECTURAL OVERVIEW .....</b>	<b>13</b>
2.1 -	BASIC CPU CONCEPTS AND OPTIMIZATIONS.....	13
2.1.1 -	High Instruction Bandwidth / Fast Execution .....	14
2.2 -	HIGH FUNCTION 8 BIT AND 16 BIT ALU .....	15
2.2.1 -	Extended Bit Processing and Peripheral Control .....	15
2.2.2 -	High Performance Branch, Call, and Loop Processing .....	15
2.2.3 -	Consistent and Optimized Instruction Formats.....	16
2.2.4 -	Programmable Multiple Priority Interrupt System .....	16
2.3 -	ON-CHIP SYSTEM RESOURCES.....	17
2.3.1 -	Peripheral Event Control and Interrupt Control .....	17
2.3.2 -	Memory Areas .....	17
2.3.3 -	External Bus Interface .....	18
2.4 -	CLOCK GENERATOR .....	18
2.4.1 -	PLL Operation .....	20
2.4.2 -	Prescaler Operation.....	20
2.4.3 -	Direct Drive.....	20
2.4.4 -	Oscillator Watchdog (OWD) .....	20
2.5 -	ON-CHIP PERIPHERAL BLOCKS .....	21
2.5.1 -	Peripheral Interfaces .....	21
2.5.2 -	Peripheral Timing .....	21
2.5.3 -	Programming Hints.....	21
2.5.4 -	Parallel Ports .....	22
2.5.5 -	Serial Channels .....	22
2.5.6 -	The on-chip CAN Module .....	22
2.5.7 -	General Purpose Timer (GPT) Unit .....	23
2.5.8 -	Watchdog Timer .....	23
2.5.9 -	Capture / Compare (CAPCOM) Units .....	23
2.5.10 -	Pulse Width Modulation Unit .....	24
2.5.11 -	A/D Converter .....	24
2.6 -	PROTECTED BITS .....	25
<b>3 -</b>	<b>MEMORY ORGANIZATION.....</b>	<b>26</b>
3.1 -	INTERNAL ROM .....	27
3.2 -	INTERNAL RAM AND SFR AREA .....	28
3.2.1 -	System Stack .....	29
3.2.2 -	General Purpose Registers .....	30
3.2.3 -	PEC Source and Destination Pointers.....	31
3.2.4 -	Special Function Registers.....	31
3.3 -	THE ON-CHIP XRAM .....	32
3.3.1 -	XRAM Access Via External Masters .....	32
3.4 -	EXTERNAL MEMORY SPACE .....	33

3.5 -	CROSSING MEMORY BOUNDARIES .....	33
<b>4 -</b>	<b>THE CENTRAL PROCESSING UNIT (CPU).....</b>	<b>34</b>
4.1 -	INSTRUCTION PIPELINES .....	35
4.1.1 -	Sequential Instruction Processing .....	36
4.1.2 -	Standard Branch Instruction Processing .....	36
4.1.3 -	Cache Jump Instruction Processing .....	36
4.1.4 -	Particular Pipeline Effects.....	37
4.2 -	BIT-HANDLING AND BIT-PROTECTION .....	40
4.3 -	INSTRUCTION EXECUTION TIMES .....	41
4.4 -	CPU SPECIAL FUNCTION REGISTERS .....	41
4.4.1 -	The System Configuration Register SYSCON .....	42
4.4.2 -	The Processor Status Word PSW .....	43
4.4.3 -	The Instruction Pointer IP .....	46
4.4.4 -	The Code Segment Pointer CSP .....	46
4.4.5 -	The Data Page Pointers DPP0, DPP1, DPP2, DPP3 .....	48
4.4.6 -	The Context Pointer CP .....	49
4.4.7 -	The Stack Pointer SP .....	51
4.4.8 -	The Stack Overflow Pointer STKOV .....	51
4.4.9 -	The Stack Underflow Pointer STKUN .....	52
4.4.10 -	The Multiply / Divide High Register MDH .....	52
4.4.11 -	The Multiply / Divide Low Register MDL .....	53
4.4.12 -	The Multiply / Divide Control Register MDC .....	53
4.4.13 -	The Constant Zeros Register ZEROS .....	54
4.4.14 -	The Constant Ones Register ONES .....	54
4.4.15 -	Example.....	54
<b>5 -</b>	<b>INTERRUPT AND TRAP FUNCTIONS .....</b>	<b>55</b>
5.1 -	INTERRUPT SYSTEM STRUCTURE .....	55
5.1.1 -	Normal Interrupt Processing and PEC Service .....	58
5.1.2 -	Interrupt System Register Description.....	59
5.1.3 -	Interrupt Control Registers .....	59
5.1.4 -	Interrupt Priority Level and Group Level .....	60
5.1.5 -	Interrupt Control Functions in the PSW .....	61
5.2 -	OPERATION OF THE PEC CHANNELS .....	62
5.3 -	PRIORITIZING INTERRUPT & PEC SERVICE REQUESTS .....	64
5.3.1 -	Enabling and Disabling Interrupt Requests .....	64
5.3.2 -	Interrupt Class Management .....	64
5.4 -	SAVING THE STATUS DURING INTERRUPT SERVICE .....	65
5.4.1 -	Context Switching .....	66
5.5 -	INTERRUPT RESPONSE TIMES .....	66
5.5.1 -	PEC Response Times .....	67
5.6 -	EXTERNAL INTERRUPTS .....	69
5.6.1 -	Fast External Interrupts .....	70
5.7 -	TRAP FUNCTIONS .....	71
5.7.1 -	Software Traps .....	71
5.7.2 -	Hardware Traps .....	71

5.7.3 -	External NMI Trap .....	73
5.7.4 -	Stack Overflow Trap .....	73
5.7.5 -	Stack Underflow Trap .....	73
5.7.6 -	Undefined Opcode Trap .....	73
5.7.7 -	Protection Fault Trap .....	73
5.7.8 -	Illegal Word Operand Access Trap.....	73
5.7.9 -	Illegal Instruction Access Trap.....	73
5.7.10 -	Illegal External Bus Access Trap.....	73
<b>6 -</b>	<b>PARALLEL PORTS .....</b>	<b>74</b>
6.1 -	INTRODUCTION .....	74
6.1.1 -	Open Drain Mode .....	74
6.1.2 -	Input Threshold Control .....	75
6.1.3 -	Alternate Port Functions .....	76
6.2 -	PORT0.....	77
6.2.1 -	Alternate Functions of PORT0.....	78
6.3 -	PORT1.....	80
6.3.1 -	Alternate Functions of PORT1.....	80
6.4 -	PORT2.....	82
6.4.1 -	Alternate Functions of Port2 .....	83
6.5 -	PORT3.....	85
6.5.1 -	Alternate Functions of Port3 .....	86
6.6 -	PORT4.....	88
6.6.1 -	Alternate Functions of Port4 .....	89
6.7 -	PORT5.....	90
6.7.1 -	Alternate Functions of Port5 .....	91
6.8 -	PORT6.....	92
6.8.1 -	Alternate Functions of Port6 .....	93
6.9 -	PORT7.....	95
6.9.1 -	Alternate Functions of Port7 .....	96
6.10 -	PORT8.....	99
6.10.1 -	Alternate Functions of Port8 .....	100
<b>7 -</b>	<b>DEDICATED PINS .....</b>	<b>102</b>
<b>8 -</b>	<b>THE EXTERNAL BUS INTERFACE.....</b>	<b>104</b>
8.1 -	SINGLE CHIP MODE .....	104
8.2 -	EXTERNAL BUS MODES .....	106
8.2.1 -	Multiplexed Bus Modes .....	106
8.2.2 -	Demultiplexed Bus Modes .....	107
8.2.3 -	Switching Between the Bus Modes .....	108
8.2.4 -	External Data Bus Width .....	109
8.2.5 -	Disable / Enable Control for Pin BHE (BYTDIS) .....	110
8.2.6 -	Segment Address Generation .....	110

8.2.7 -	CS Signal Generation .....	110
8.2.8 -	Segment Address Versus Chip Select .....	111
8.3 -	<b>PROGRAMMABLE BUS CHARACTERISTICS .....</b>	<b>111</b>
8.3.1 -	ALE Length Control .....	112
8.3.2 -	Programmable Memory Cycle Time .....	113
8.3.3 -	Programmable Memory Tri-state Time .....	113
8.3.4 -	Read / Write Signal Delay .....	114
8.3.5 -	READY Polarity .....	115
8.3.6 -	READY / $\overline{\text{READY}}$ Controlled Bus Cycles .....	115
8.3.7 -	Programmable Chip Select Timing Control .....	116
8.4 -	<b>CONTROLLING THE EXTERNAL BUS CONTROLLER .....</b>	<b>117</b>
8.4.1 -	Definition of Address Areas .....	121
8.4.2 -	Address Window Arbitration .....	122
8.4.3 -	Precautions and Hints .....	123
8.5 -	<b>EBC IDLE STATE .....</b>	<b>123</b>
8.6 -	<b>EXTERNAL BUS ARBITRATION .....</b>	<b>124</b>
8.6.1 -	Connecting Bus Masters .....	125
8.6.2 -	Entering the Hold State .....	125
8.6.3 -	Exiting the Hold State .....	126
8.7 -	<b>THE XBUS INTERFACE .....</b>	<b>127</b>
<b>9 -</b>	<b>THE GENERAL PURPOSE TIMER UNITS .....</b>	<b>128</b>
9.1 -	TIMER BLOCK GPT1 .....	128
9.1.1 -	GPT1 Core Timer T3 .....	130
9.1.2 -	GPT1 Auxiliary Timers T2 and T4 .....	137
9.1.3 -	Interrupt Control for GPT1 Timers .....	142
9.2 -	TIMER BLOCK GPT2 .....	143
9.2.1 -	GPT2 Core Timer T6 .....	145
9.2.2 -	Interrupt Control for GPT2 Timers and CAPREL .....	154
<b>10 -</b>	<b>ASYNCHRONOUS/SYNCHRONOUS SERIAL INTERFACE .....</b>	<b>155</b>
10.1 -	ASYNCHRONOUS OPERATION .....	158
10.2 -	SYNCHRONOUS OPERATION .....	161
10.3 -	HARDWARE ERROR DETECTION .....	162
10.4 -	ASC0 BAUD RATE GENERATION .....	162
10.5 -	ASC0 INTERRUPT CONTROL .....	163
<b>11 -</b>	<b>HIGH-SPEED SYNCHRONOUS SERIAL INTERFACE .....</b>	<b>165</b>
11.1 -	FULL-DUPLEX OPERATION .....	170
11.2 -	HALF DUPLEX OPERATION .....	172
11.2.1 -	Port Control .....	173
11.3 -	BAUD RATE GENERATION .....	174
11.4 -	ERROR DETECTION MECHANISMS .....	174

---

11.5 -	SSC INTERRUPT CONTROL .....	175
<b>12 -</b>	<b>WATCHDOG TIMER .....</b>	<b>177</b>
12.1 -	OPERATION OF THE WATCHDOG TIMER.....	177
<b>13 -</b>	<b>BOOTSTRAP LOADER .....</b>	<b>179</b>
<b>14 -</b>	<b>THE CAPTURE / COMPARE UNITS.....</b>	<b>183</b>
14.1 -	CAPCOM TIMERS .....	186
14.2 -	CAPCOM UNIT TIMER INTERRUPTS .....	188
14.3 -	CAPTURE / COMPARE REGISTERS .....	189
14.3.1 -	Selection of Capture Modes and Compare Modes .....	191
14.4 -	CAPTURE MODE .....	191
14.5 -	COMPARE MODES .....	192
14.5.1 -	Compare Mode 0.....	193
14.5.2 -	Compare Mode 1 .....	194
14.5.3 -	Compare Mode 2.....	195
14.5.4 -	Compare Mode 3.....	196
14.5.5 -	Double Register Compare Mode .....	196
14.6 -	CAPTURE / COMPARE INTERRUPTS .....	198
<b>15 -</b>	<b>PULSE WIDTH MODULATION MODULE .....</b>	<b>200</b>
15.1 -	OPERATING MODES .....	202
15.1.1 -	Mode 0: Standard PWM Generation (Edge Aligned PWM) .....	202
15.1.2 -	Mode 1: Symmetrical PWM Generation (Center Aligned PWM) .....	204
15.1.3 -	Burst Mode .....	205
15.1.4 -	Single Shot Mode .....	206
15.2 -	PWM MODULE REGISTERS.....	207
15.3 -	INTERRUPT REQUEST GENERATION.....	209
15.4 -	PWM OUTPUT SIGNALS .....	210
<b>16 -</b>	<b>ANALOG / DIGITAL CONVERTER .....</b>	<b>211</b>
16.1 -	MODE SELECTION AND OPERATION.....	212
16.1.1 -	Fixed Channel Conversion Modes .....	214
16.1.2 -	Auto Scan Conversion Modes .....	214
16.1.3 -	Wait for ADDAT Read Mode .....	215
16.1.4 -	Channel Injection Mode.....	216
16.2 -	CONVERSION TIMING CONTROL .....	218
16.3 -	CALIBRATION.....	218
16.4 -	A/D CONVERTER INTERRUPT CONTROL.....	218
<b>17 -</b>	<b>ON-CHIP CAN INTERFACE .....</b>	<b>220</b>
17.1 -	THE CAN CONTROLLER .....	220

17.2 -	REGISTER AND MESSAGE OBJECT ORGANIZATION .....	222
17.3 -	CAN INTERRUPT HANDLING .....	225
17.4 -	THE MESSAGE OBJECT .....	228
17.5 -	ARBITRATION REGISTERS.....	230
17.6 -	INITIALIZATION AND RESET.....	239
17.7 -	CAN APPLICATION INTERFACE .....	240
<b>18 -</b>	<b>SYSTEM RESET .....</b>	<b>241</b>
18.1 -	TYPES OF RESET .....	242
18.1.1 -	ST10F167 Synchronous Hardware Reset.....	242
18.1.2 -	ST10C167 - ST10R167 Synchronous Hardware Reset.....	242
18.1.3 -	Asynchronous Hardware Reset.....	243
18.1.4 -	Software Reset.....	243
18.1.5 -	Watchdog Timer Reset.....	243
18.1.6 -	Bi-Directional Reset.....	243
18.2 -	PINS AFTER RESET .....	244
18.2.1 -	System Start-up Configuration .....	247
<b>19 -</b>	<b>POWER REDUCTION MODES .....</b>	<b>252</b>
19.1 -	IDLE MODE .....	252
19.2 -	POWER DOWN MODE.....	253
19.2.1 -	Protected Power Down Mode.....	253
19.3 -	INTERRUPTIBLE POWER DOWN MODE .....	253
19.4 -	OUTPUT PIN STATUS.....	256
<b>20 -</b>	<b>REGISTER SET .....</b>	<b>257</b>
20.1 -	REGISTER DESCRIPTION FORMAT .....	257
20.2 -	GENERAL PURPOSE REGISTERS (GPRS) .....	257
20.3 -	SPECIAL FUNCTION REGISTERS ORDERED BY NAME .....	259
20.4 -	REGISTERS ORDERED BY ADDRESS .....	265
20.5 -	SPECIAL NOTES.....	271
20.6 -	IDENTIFICATION REGISTERS .....	271
<b>21 -</b>	<b>SYSTEM PROGRAMMING.....</b>	<b>273</b>
21.1 -	STACK OPERATIONS .....	275
21.2 -	REGISTER BANKING .....	278
21.3 -	PROCEDURE CALL ENTRY AND EXIT.....	278
21.4 -	TABLE SEARCHING.....	280
21.5 -	PERIPHERAL CONTROL AND INTERFACE .....	280
21.6 -	FLOATING POINT SUPPORT .....	280



---

21.7 -	TRAP / INTERRUPT ENTRY AND EXIT.....	281
21.8 -	INSEPARABLE INSTRUCTION SEQUENCES .....	281
21.9 -	OVERRIDING THE DPP ADDRESSING MECHANISM .....	281
21.10 -	HANDLING THE INTERNAL ROM .....	282
21.11 -	PITS, TRAPS AND MINES.....	283
<b>22 -</b>	<b>KEY WORD INDEX.....</b>	<b>284</b>
<b>23 -</b>	<b>INDEX OF REGISTERS.....</b>	<b>288</b>
<b>24 -</b>	<b>REVISION HISTORY .....</b>	<b>291</b>
24.1 -	REVISION OF THE 28TH OF AUGUST 2000 .....	291
24.2 -	REVISION OF THE 7TH OF AUGUST 2002 .....	292



### 1 - INTRODUCTION

This manual describes the functionality of the ST10X167 group of devices. ST10X167 is a generic term covering the ROMless ST10R167, ROM ST10C167 and flash ST10F167 devices.

The ST10R167 and ST10C167 are functionally equivalent, apart from the memory arrangement. The ST10F167 is an older derivative of the device and has different functionality. Section 1.1 - "Differences Between the ST10R167, ST10C167 & ST10F167" details the functional differences between the ST10X167 derivatives.

Comments have been made throughout the manual to highlight differences where applicable.

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 ST10F167 flash configurations are not discussed in this manual. Refer to the ST10F167 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 21 - 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 ST10X167 software and hardware development tools include:

- Compilers (C, FORTH, C++)
- Macro-Assemblers, Linkers, Locators, Library Managers, Format-Converters
- HLL debuggers
- Real-Time operating systems
- In-Circuit Emulators (based on bondout or standard chips)
- Plug-In emulators
- Emulation and Clip-Over adapters, production sockets
- Logic Analyzer disassemblers
- Evaluation Boards with monitor programs
- Industrial boards (also for CAN, FUZZY, PROFIBUS, FORTH applications)
- Network driver software (CAN, PROFIBUS)

### 1.1 - Differences Between the ST10R167, ST10C167 & ST10F167

Apart from the memory arrangement, the ST10R167 and ST10C167 are functionally the same as each other. The ST10F167 is an older derivative of the device and has different functionality. This section summarizes the

functional differences between the ST10R167, ST10C167 devices and the older ST10F167. These differences have been broken into functional groups.

The differences have been highlighted again in the individual chapters of the user manual where applicable.

Feature	ST10C167 & ST10R167	ST10F167
<b>Power Reduction</b> <ul style="list-style-type: none"> <li>– Interruptible power down mode</li> <li>– Return from powerdown mode: Selected by setting the bit PWDCFG in the SYSCON register to '1' and uses the VPP/RPD pin for C167/R167. For ST10F167 the VPP/RPD pin is used for the flash programming voltage.</li> </ul>	Available Available	Not available Not available
<b>External Bus Interface</b> <ul style="list-style-type: none"> <li>– Programmable chip select timing control.</li> <li>– READY polarity: The active level of the READY pin can be selected by software</li> <li>– Address window arbitration: For each address access the EBC compares the current address with all address select register.</li> </ul>	Available Available Available	Not available Not available Not available
<b>General Purpose Timers</b> <ul style="list-style-type: none"> <li>– Incremental interface mode</li> </ul>	Available	Not available
<b>Syscon Register</b> Bit allocation <ul style="list-style-type: none"> <li>– XPEN - XBUS Peripheral Enable bit enables XRAM and XCAN. For ST10F167 this bit is used to enable XRAM only as XCAN is always enabled.</li> <li>– BDRSTEN - Bidirectional reset enable</li> <li>– OWDDIS - Oscillator watchdog disable control</li> <li>– PWDCFG - Power down mode configuration control</li> <li>– CSCFG - Chip select configuration control.</li> </ul>	Allocated Allocated Allocated Allocated	Allocated Not allocated Not allocated Not allocated
<b>System reset</b> <ul style="list-style-type: none"> <li>– Bi-directional reset</li> <li>– Asynchronous reset function</li> </ul>	Available Available	Not available Not available
<b>CAN module</b> <ul style="list-style-type: none"> <li>– Optional disabling of the CAN module by the XPEN bit in the SYSCON register.</li> </ul>	Available	Not available
<b>Clock generation</b> <ul style="list-style-type: none"> <li>– Optional disabling of the oscillator watchdog by selection of the OWDDIS bit in the SYSCON register.</li> </ul>	Available	Not available
<b>Dedicated pins</b> <ul style="list-style-type: none"> <li>– The active level of the READY pin can be selected by software.</li> <li>– V<sub>PP</sub> used for flash programming voltage for ST10F167 or exit from powerdown for all ST10C167 and ST10R167 devices.</li> </ul>	Available	Not available
<b>Identification registers</b> <ul style="list-style-type: none"> <li>– Four identification registers</li> </ul>	Available	Not available

**1.2 - Abbreviations**

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

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

## 2 - ARCHITECTURAL OVERVIEW

ST10X167 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 ST10X167. (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** : ST10X167 functional block diagram

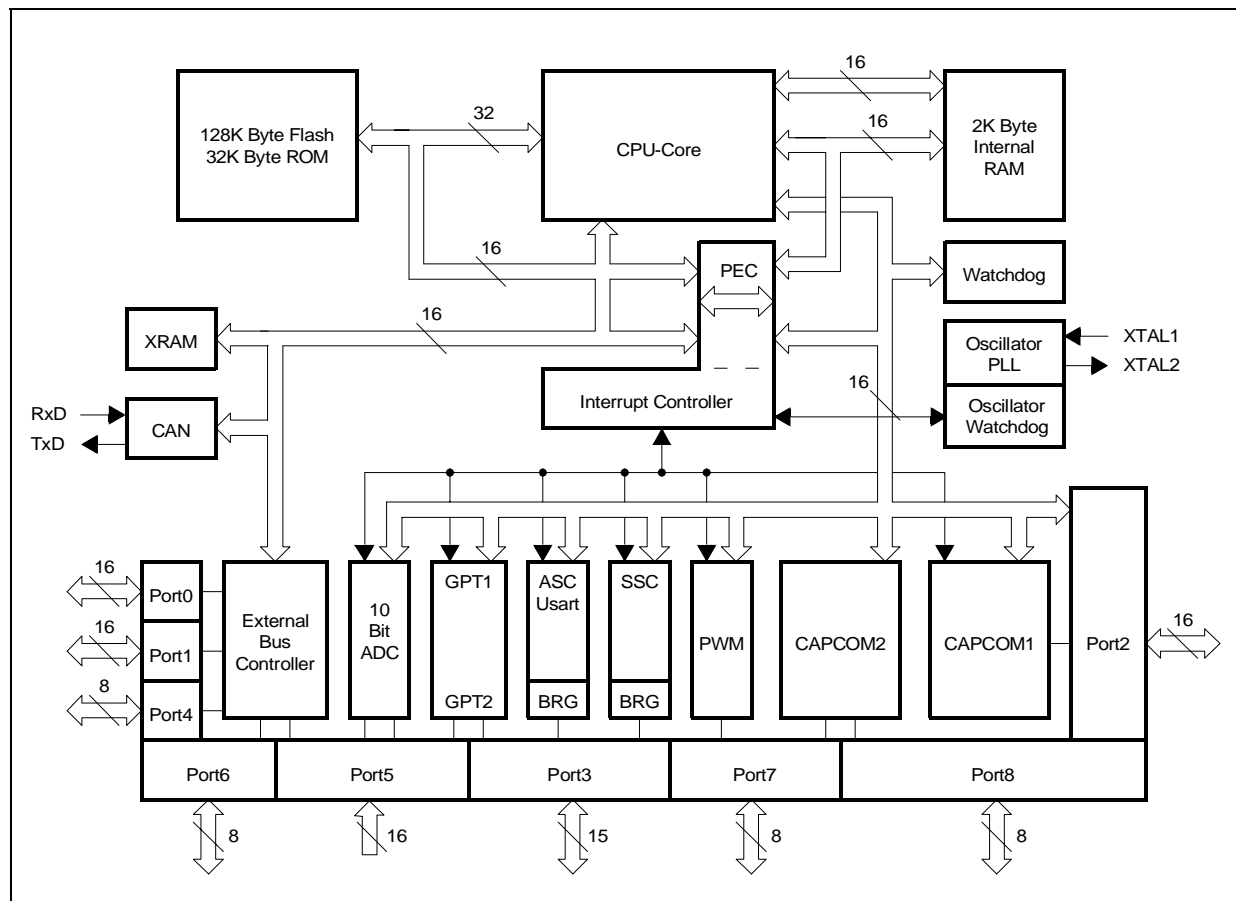
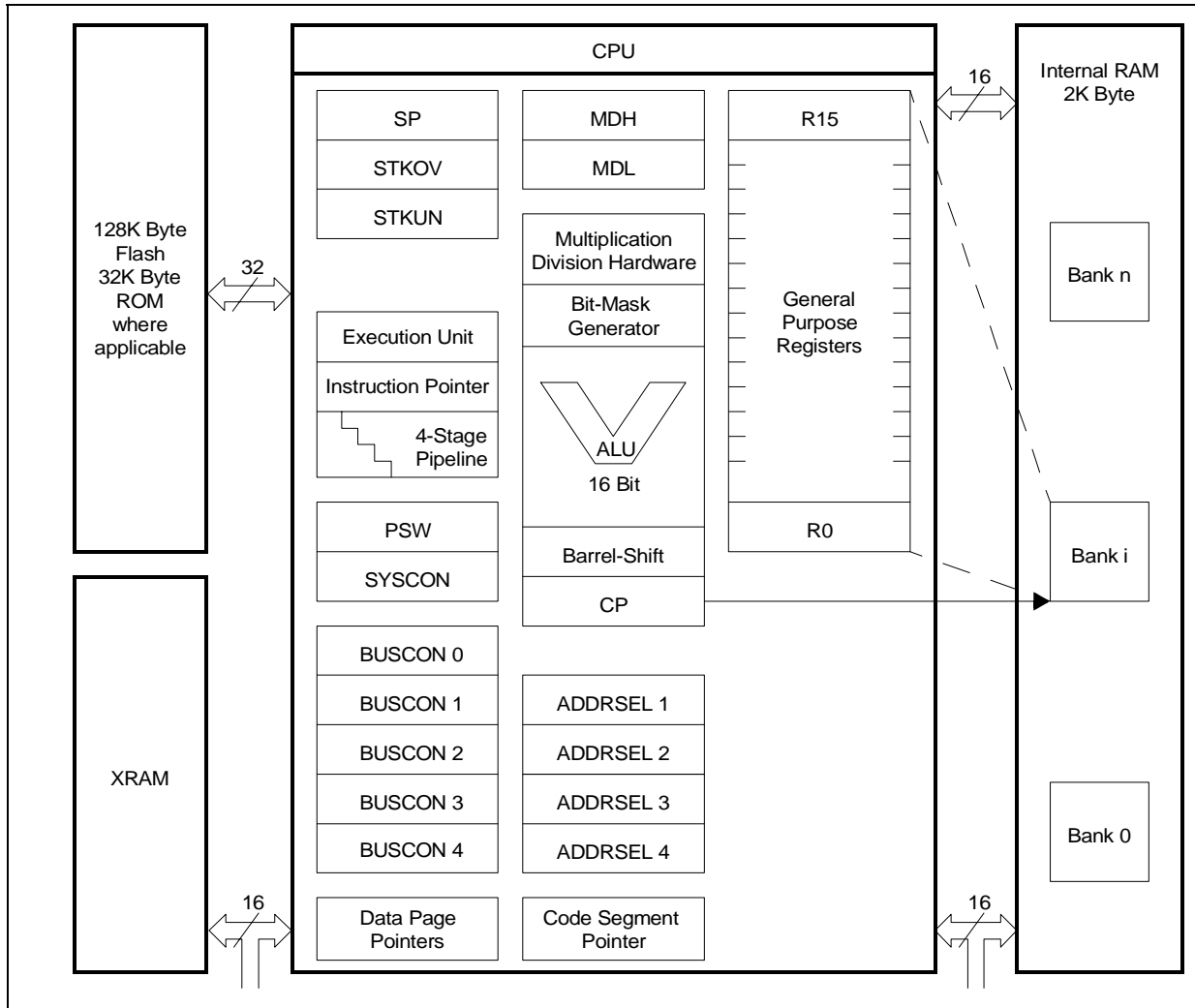


Figure 2 : CPU Block Diagram



**2.1.1 - High Instruction Bandwidth / Fast Execution**

Most of the ST10X167'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 Bit to be shifted. Multiple-cycle instructions have been optimized: branches are carried out in 2 CPU clock cycles, 16 × 16 Bit multiplication in 5 CPU clock cycles and a 32/16 Bit division in 10 CPU clock cycles. The jump cache reduces the execution time of repeatedly performed jumps in a loop, from 2 CPU clock cycles to 1 CPU clock 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 ROM or RAM or from the external memory, based on the current IP value.
- Decode: In this stage, the previously fetched instruction is decoded and the required operands are fetched.
- Execute: In this stage, the specified operation is performed on the previously fetched operands.
- Write back: In this stage, the result is written to the specified location.

If this technique 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 Bit six and seven 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 four Bit to be multiplied and two Bit 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 ST10X167 instruction set. Possible operand types are Bit, Byte and Words. Specific instruction support the conversion (extension) of Byte 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 Bitfield. Once having been accepted by the CPU, an interrupt service can only be interrupted by a higher prioritized service request. For standard interrupt processing, each of the possible interrupt sources has a dedicated vector location.
- 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 250ns to 500ns (in case of internal program execution), the ST10X167 is capable of fast reaction to non-deterministic events.



The ST10X167 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 ST10X167 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 ST10X167 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 ST10X167 is configured in a Von Neumann architecture which means that code memory, data memory, registers and I/O ports are organized within the same linear address space which covers up to 16M Byte. 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 2K Byte 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 2K Byte 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.

**An optional internal ROM** 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 ROM is the fastest of all possible alternatives.

**For Special Function Registers** 1024 Byte of the address space are reserved. The standard Special Function Register area (SFR) uses 512 Byte, while the Extended Special Function Register area (ESFR) uses the other 512 Byte. (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 ST10X167 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 16M Byte 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 / demultiplexed), the data bus width (8 Bit / 16 Bit) and even the length of a bus cycle (waitstates, signal delays) can be selected independently.

This allows 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, demultiplexed.
- 16-/18-/20-/24 Bit Addresses, and 8 Bit data, demultiplexed.
- 16-/18-/20-/24 Bit Addresses, and 16 Bit data, multiplexed.
- 16-/18-/20-/24 Bit Addresses, and 8 Bit data, multiplexed.

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

Important timing characteristics of the external bus interface (waitstates, ALE length and Read/Write Delay) have been made programmable to

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 64K Byte 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-Module are examples for these X-Peripherals.

### 2.4 - Clock Generator

The on-chip clock generator provides the ST10X167 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 7 - 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.

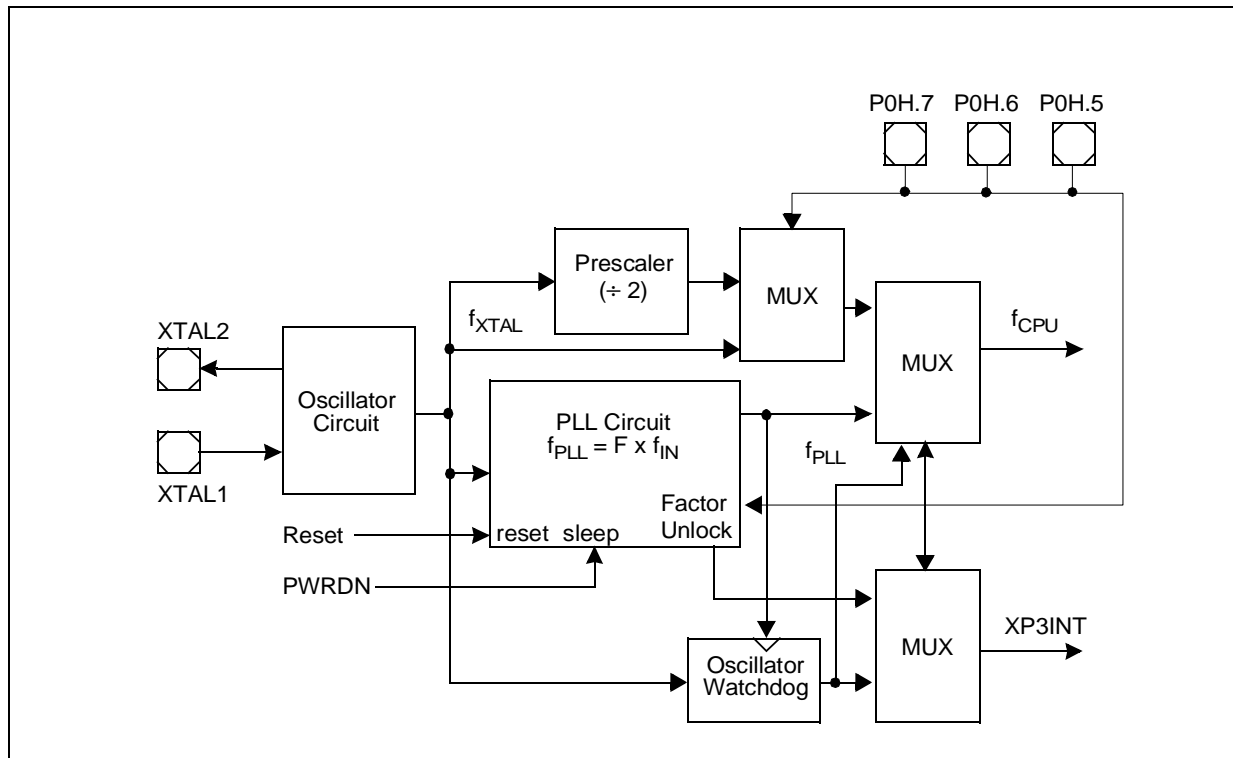
While the CPU clock is stopped during idle mode, the peripheral clock keeps running. Both clocks are switched off when the power-down mode is entered.

The on-chip PLL circuit allows operation of the ST10X167 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 12 - 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

P0.15-13 (P0H.7-5)	ST10C167 & ST10R167 CPU Frequency $f_{CPU} = f_{XTAL} \times F$	ST10F167 CPU Frequency $f_{CPU} = f_{XTAL} \times F$	Notes
1 1 1	$f_{XTAL} \times 4$	$f_{XTAL} \times 4$	Default configuration
1 1 0	$f_{XTAL} \times 3$	$f_{XTAL} \times 4$	
1 0 1	$f_{XTAL} \times 2$	$f_{XTAL} \times 4$	
1 0 0	$f_{XTAL} \times 5$	$f_{XTAL} \times 4$	
0 1 1	$f_{XTAL} \times 1$	$f_{XTAL} \times 1$	Direct drive
0 1 0	$f_{XTAL} \times 1.5$	$f_{XTAL} \times 1$	
0 0 1	$f_{XTAL} / 2$	$f_{XTAL} \times 1$	CPU clock via prescaler <sup>1</sup>
0 0 0	$f_{XTAL} \times 2.5$	$f_{XTAL} \times 1$	

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

### 2.4.1 - 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<sub>DD</sub> 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...5 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  $F \times f_{XTAL}$ , and the PLL locks to the external clock.

**Note** If the ST10X167 is required to operate on the desired CPU clock directly after reset, make sure that  $\overline{RSTIN}$  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...5 MHz). The basic frequency is still generated and allows the CPU to execute emergency actions in case of a loss of the external clock.

### 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...5 MHz, and delivers the clock signal for the Oscillator Watchdog, except for ST10C167 or ST10R167 where 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...5 MHz, and delivers the clock signal for the Oscillator Watchdog, except for ST10C167 or ST10R167 where if Bit OWDDIS is set the PLL is switched off.

### 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:

- For the ST10F167, the oscillator watchdog is always enabled
- For the ST10C167 and ST10R167, the oscillator watchdog is enabled by default after reset. To disable the OWD, set bit OWDDIS of the SYSCON register..
- When the OWD is enabled, the PLL runs on its free-running frequency, and increments the Oscillator Watchdog counter.
- On each transition of XTAL1 pin, the Oscillator Watchdog is cleared.

If an external clock failure occurs, then the Oscillator Watchdog counter overflows (after 16 PLL clock cycles). The CPU clock signal will be switched to the PLL clock signal (in this case, the PLL will run on its basic frequency of 2...5 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 exists 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'FFFh), 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 ST10X167 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 4-channel Pulse Width Modulation unit,
- A 10 Bit Analog / Digital Converter,
- Nine I/O ports with a total of 111 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), EXT(S)(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 ST10X167's SFRs are marked as 'Reserved'. User software should never write '1'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 ST10X167 provides up to 111 I/O lines which are organized into eight input/output ports and one input port. All port lines are Bit-addressable, and all input/output lines are individually (Bit wise) programmable as inputs or outputs via direction registers. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs.

The output drivers of three I/O ports can be configured (pin by pin) for push-pull operation or 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 and PORT1 may be used as data and address lines respectively when accessing external memory.
- 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 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 64K Byte of memory.
- Port5 is used as analog input channels of the A/D converter or as timer control signals.
- Port6 provides optional bus arbitration signals (BREQ, HLDA, HOLD) and chip select signals.
- Port7 provides the output signals from the PWM unit and inputs/outputs for the CPACOM2 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.

#### **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 Bit. 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 Module**

The integrated CAN Module handles the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active). The 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 Byte. The Bit timing is derived from the XCLK and is programmable up to a data rate of 1M Baud. 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 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 16 multiplexed input channels and a sample and hold circuit has been integrated on-chip for analog signal measurement.

It 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 ST10X167 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 ST10X167 MCU provide 106 protected Bit. These Bit are modified by the on-chip hardware during special events like power-on reset, power failure, application hardware, etc. These bit 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 :  $\Sigma$  = 106 protected Bit.

3 - MEMORY ORGANIZATION

The memory space of the ST10X167 is configured in a Von-Neumann architecture. 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 ROM / Flash, internal RAM, the internal Special Function Register Areas (SFRs and ESRs), the address areas for integrated XBUS peripherals like XRAM or CAN module and external memory are mapped into one common address space.

The ST10X167 provides a total addressable memory space of 16M Byte. This address space is arranged as 256 segments of 64K Byte each, and each segment is again subdivided into four data pages of 16K Byte each (see Figure 4).

Most of the internal memory areas are mapped into segment 0, named system segment. The upper

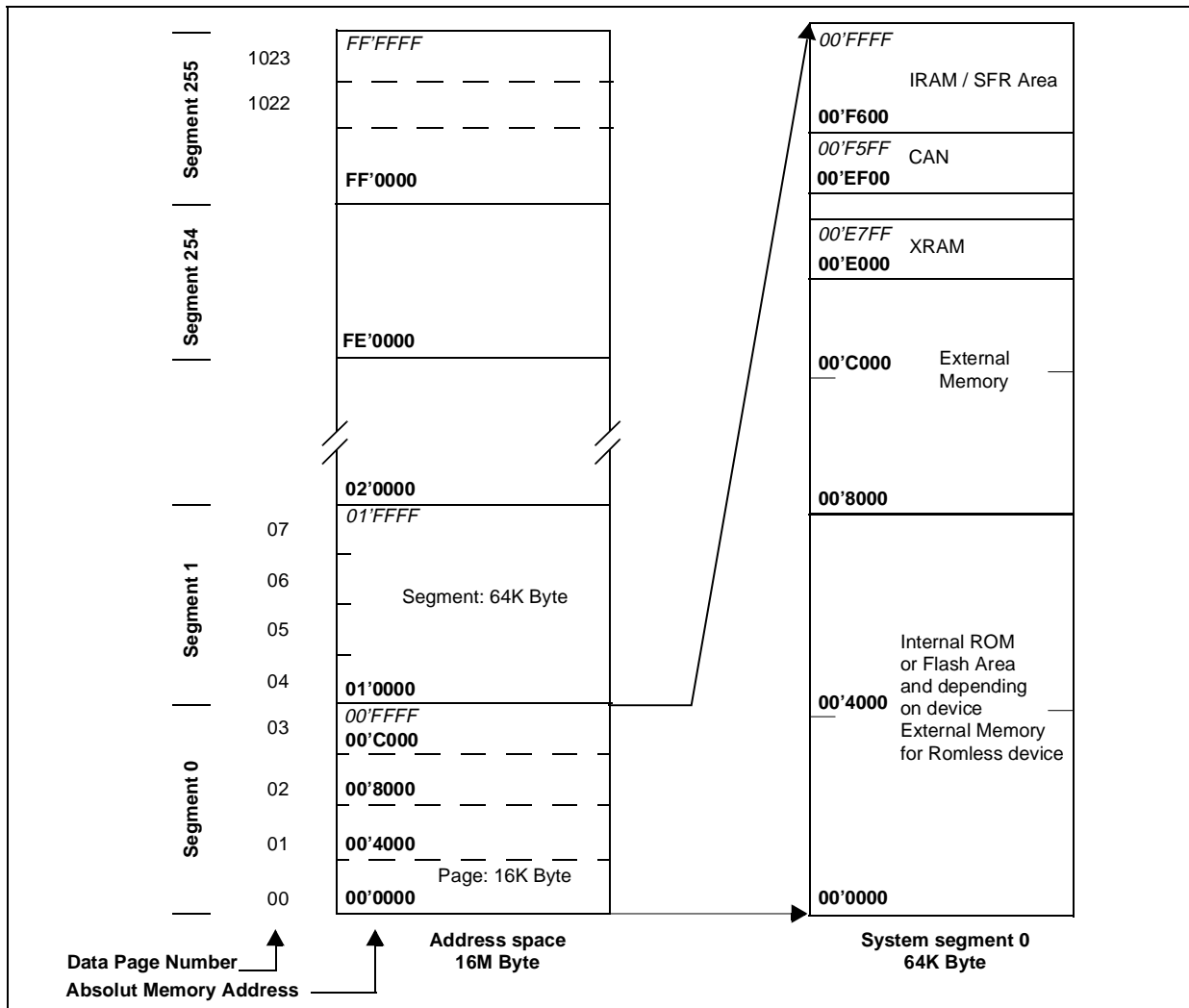
4K Byte of segment 0 (00'F000h...00'FFFFh) hold the Internal RAM and Special Function Register Areas (SFR and ESR). The lower 32K Byte of segment 0 (00'0000h...00'7FFFh) can be occupied by a part of the on-chip ROM or Flash Memory and is called the internal ROM area.

This ROM 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 ROM 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.

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

Figure 4 : Memory areas and address space

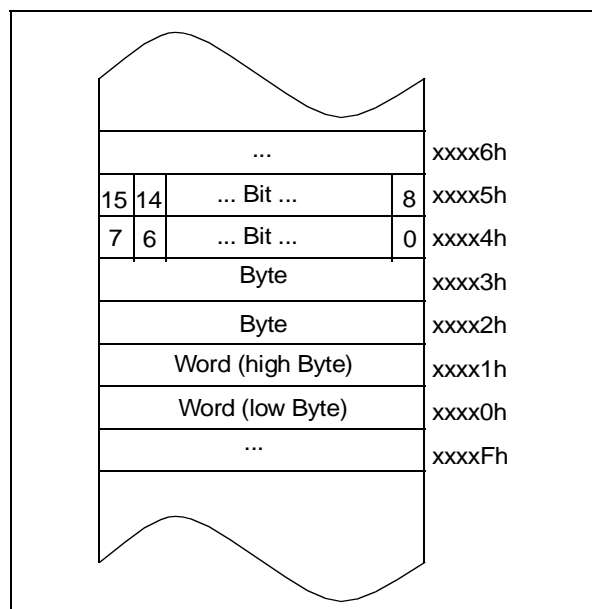


Byte are stored at even or odd Byte addresses. Words are stored in ascending memory locations with the low Byte at an even Byte address being followed by the high Byte at the next odd Byte address.

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, ROM, RAM) and organizational (page, segment) memory area.

### 3.1 - Internal ROM

The ST10X167 reserves an address area of variable size (depending on the version) for on-chip mask-programmable ROM (organized as  $X \times 32$ ) or Flash memory.

The lower 32 KByte of the on-chip ROM/Flash are referred to as "Internal ROM Area". Internal ROM accesses are globally enabled or disabled via Bit

ROMEN in register SYSCON. This Bit is set during reset according to the level on pin  $\bar{E}A$ , or may be altered via software. If enabled, the internal ROM area occupies the lower 32 KByte of either segment 0 or segment 1. This ROM mapping is controlled by Bit ROMS1 in register SYSCON.

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

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

The internal ROM/Flash can be used for both code (instructions) and data (constants, tables, etc.) storage.

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

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

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

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

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

**Note** The 'x' in the locations above depends on the available ROM/Flash memory and on the mapping.

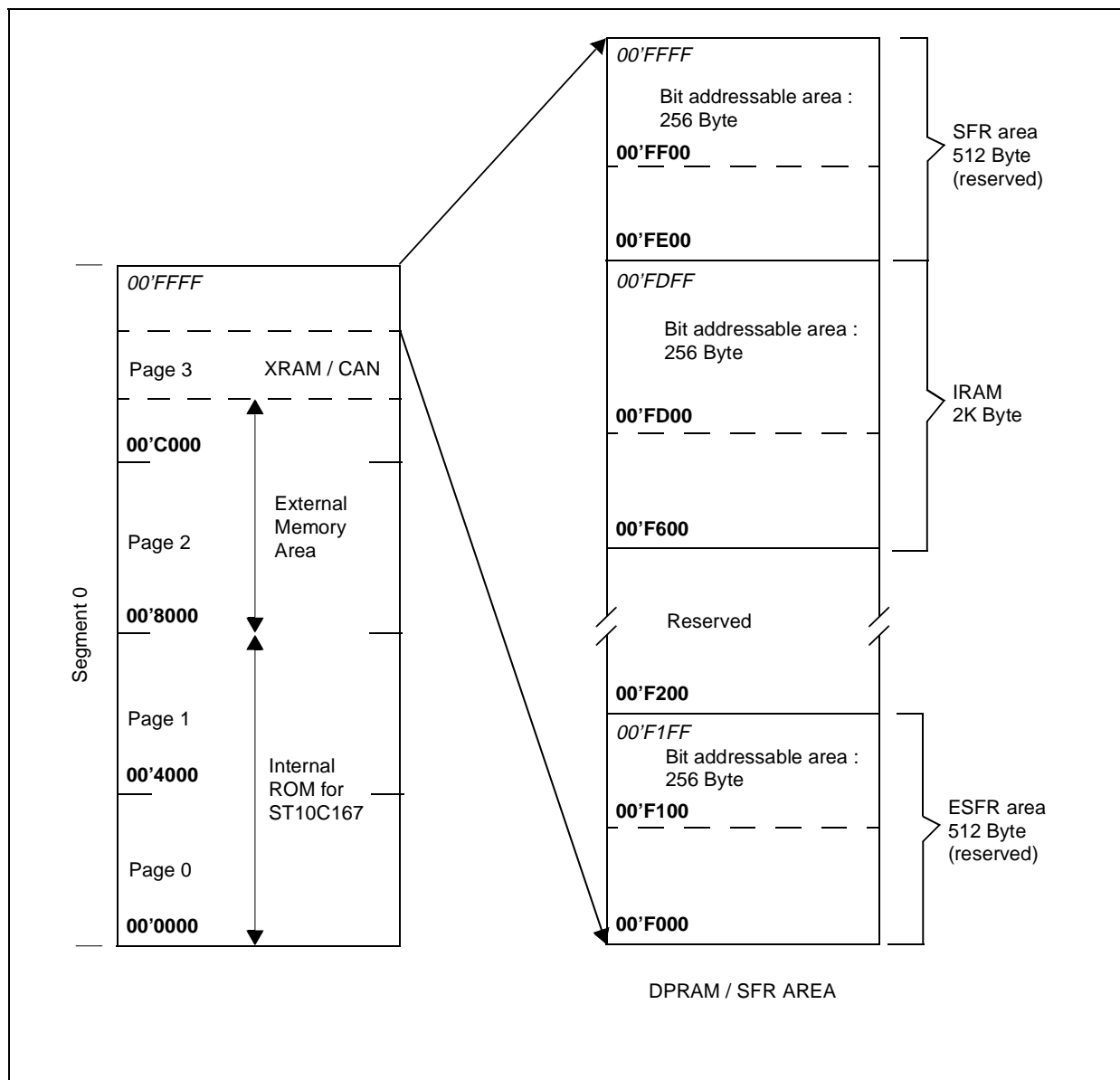
The internal ROM may be enabled, disabled or mapped into segment 0 or segment 1 under software control. "Handling the internal ROM" Section 21.10 describes the mapping procedures and precautions.

**3.2 - Internal RAM and SFR Area**

The RAM/SFR area is located within data page 3 and provides access to 2K Byte 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



- 1 The upper 256 Byte of SFR area, ESRF 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 cause erroneous results.

Any Word and Byte data in the internal RAM can be accessed via indirect or long 16Bit 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 Byte 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 Bitfield STKSZ in the SYSCON register (see Table 3).

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 2K Byte stack option).

The technique of implementing this circular stack is described in Section 21.1 Circular (virtual) Stack.

**Table 3 : 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 4).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 Byte.

The GPRs are accessed via short 2, 4 or 8 Bit addressing modes using the context pointer (CP) register as base address (independent of the current DPP register contents). Additionally, each

Bit in the currently active register bank can be accessed individually.

The ST10X167 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 21.2.

**Table 4 :** 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

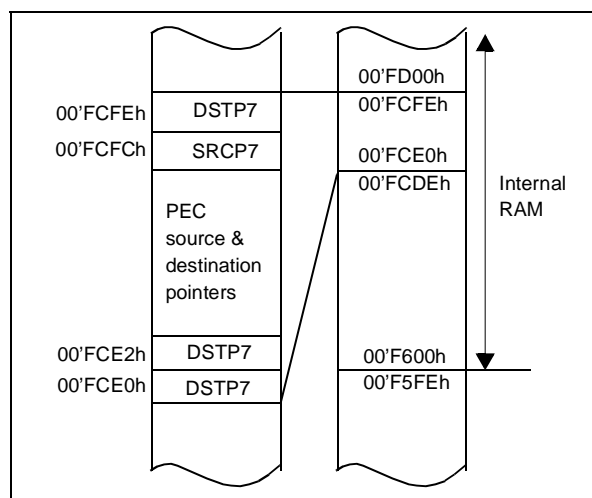
### 3.2.3 - PEC Source and Destination Pointers

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 21 - System Programming.

**Figure 7** : Location of the PEC pointers



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)

;----- ;-----
MOV     T8REL, R1        ;The scope of the EXTR #4 instruction ends here!
                               ;T8REL uses 16 Bit address, R1 is duplicated and
                               ;does not require switching

```

### 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 ST10X167 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.

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 XRAM area is located within data page 3 and provides access to 2K Byte of on-chip RAM (organized as 1K x 16). As the XRAM is connected to the internal XBUS it is accessed like external memory, however, no external bus cycles are executed for these accesses.

XRAM accesses are globally enabled or disabled via Bit XPEN in the SYSCON register. This Bit is cleared after reset and may be set via software during the initialization to allow accesses to the on-chip XRAM. When the XRAM is disabled (default after reset) all accesses to the XRAM area are mapped to external locations. The XRAM may be used for both code (instructions) and data (variables, user stack, tables, etc.) storage.

Code fetches are always made on even Byte addresses. The highest possible code storage location in the XRAM is either 00'E7FEh for single Word instructions, or 00'E7FCh for double Word instructions.

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

Any Word and Byte data read accesses may use the indirect or long 16Bit 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'E7FEh. For PEC data transfers the XRAM can be accessed independently of the contents of the DPP registers, via the PEC source and destination pointers.

Note For the ST10C167 / ST10R167, the XPEN Bit in the SYSCON register is used to enable or disable the CAN. For the ST10F167, the CAN is always enabled. As the XRAM appears like external memory it cannot be used for the ST10X167's system stack or register banks. The

XRAM is not provided for single Bit storage and therefore is not Bit addressable.

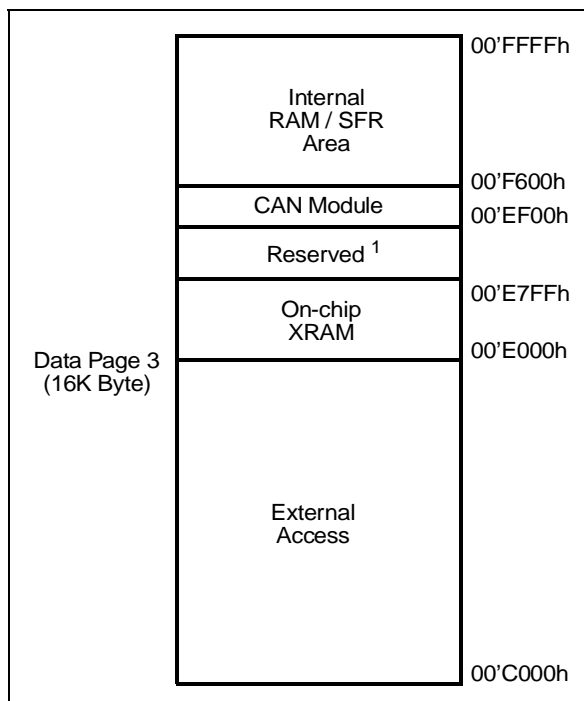
The on-chip XRAM is accessed without any waitstates, using 16 Bit demultiplexed bus cycles which takes one instruction cycle. Even if the XRAM is used as external memory, it does not occupy BUSCONx / ADDRSELx registers, but is selected via additional dedicated XBCON / XADRS registers. These registers are mask-programmed and are not user accessible. With these registers the address area 00'E000<sub>H</sub> to 00'E7FF<sub>H</sub> is reserved for XRAM accesses.

#### 3.3.1 - XRAM Access Via External Masters

When Bit XPER-SHARE in register SYSCON is set the on-chip XRAM of the ST10X167 can be accessed by an external master during hold mode, via the ST10X167's bus interface. These external accesses must use the same configuration as the internally programmed, (demultiplexed bus, 100 ns minimum access cycle time). No waitstates 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'E800h to 00'EEFFh is mapped to external memory but should be reserved for reasons of upward compatibility.



### 3.4 - External Memory Space

The ST10X167 is capable of using an address space of up to 16M Byte. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (Flash, ROM or RAM) or for registers, may refer to external memory locations. This external memory is accessed via the ST10X167's external bus interface.

**Four memory bank sizes** are supported:

- Non-segmented mode: 64K Byte with A15...A0 on PORT0 or PORT1
- 2 Bit segmented mode: 256K Byte with A17...A16 on Port4 and A15...A0 on PORT0 or PORT1
- 4 Bit segmented mode: 1M Byte with A19...A16 on Port4 and A15...A0 on PORT0 or PORT1
- 8 Bit segmented mode: 16M Byte 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 ST10X167 also supports **four different bus types**:

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

Memory model and bus mode are selected during reset by pin  $\bar{E}A$  and PORT0 pins. For further details about the external bus configuration and control, (See Chapter 8 - 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 ST10X167 is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

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

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

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

**Segments** are contiguous blocks of 64K Byte 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 16K Byte 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 16K Byte 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 ST10X167 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 8 - The External Bus Interface).

The on-chip peripheral units of the ST10X167 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:

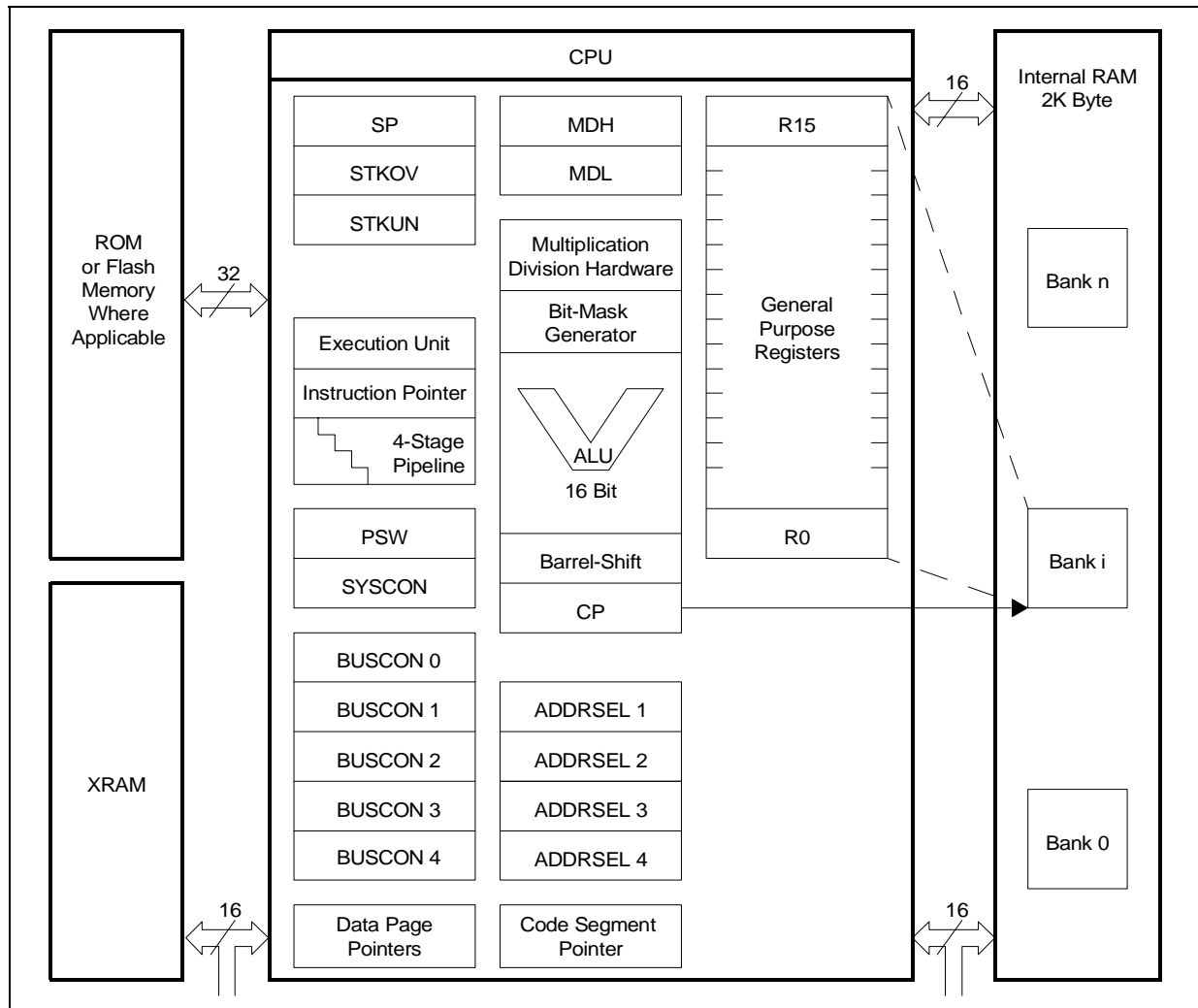
- **Reset state:** Any reset (hardware, software, watchdog) forces the CPU into a predefined active state.
- **IDLE state:** The clock signal to the CPU itself is switched off, while the clocks for the on-chip peripherals keep running.
- **POWER DOWN state:** All of the on-chip clocks are switched off.

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 ST10X167 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:

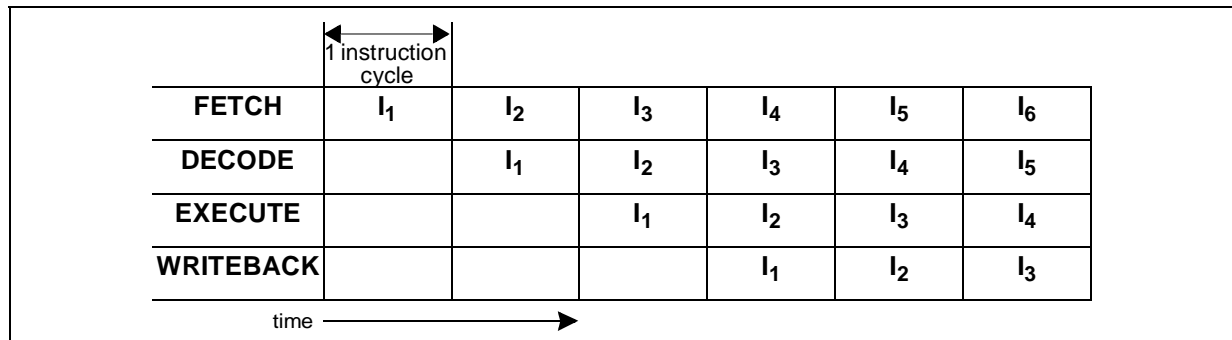
- Fetch: An instruction selected by the Instruction Pointer (IP) and the Code Segment Pointer (CSP) is fetched from either the internal memory, internal RAM, or external memory.
- Decode: Instructions are decoded and, if required, the operand addresses are calculated and the respective operands are fetched. 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 ST10X167 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

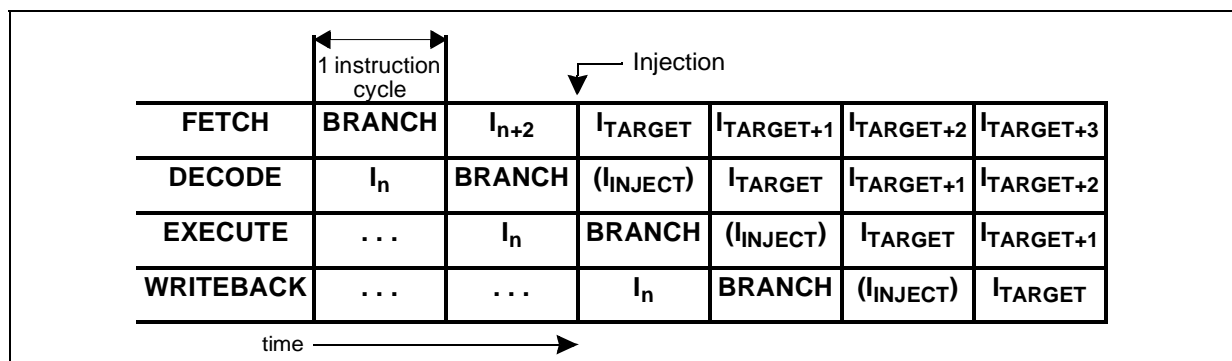
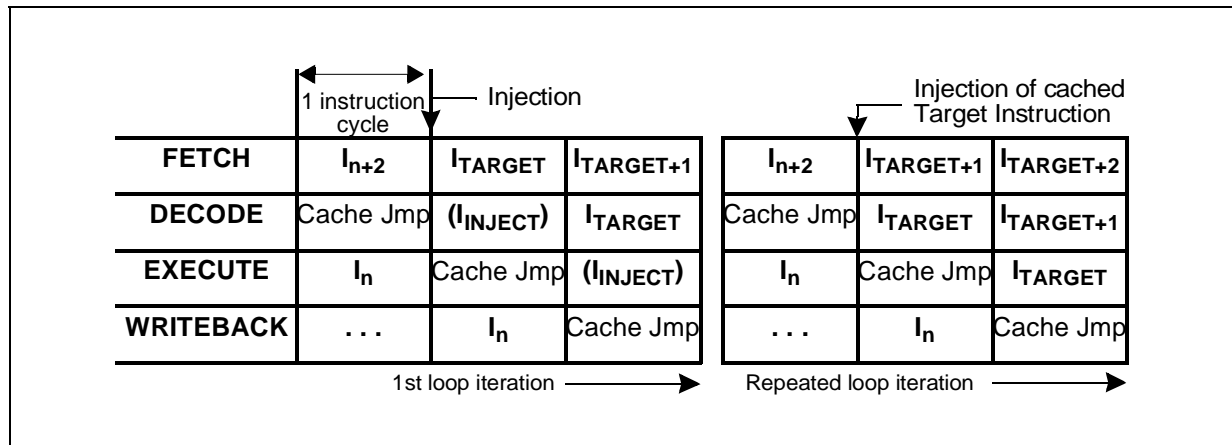


Figure 12 : Cache jump instruction pipelining



#### 4.1.4 - Particular Pipeline Effects

Since up to four different instructions are processed simultaneously, additional hardware has been included in the ST10X167 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 ST10X167.

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
           IN-1             ; non-critical instruction
CRIT_1ST:  IN               ; start of non-interruptible critical
           ; sequence
           . . .
CRIT_LAST: IN+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 resources.

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 ST10X167 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 Byte 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 5 shows the minimum execution times required to process a ST10X167 instruction fetched from the internal ROM, 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 ST10X167 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 (waitstates). The operand and instruction accesses listed below can extend the execution time of an instruction:

- Internal ROM / 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 ROM / 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 5** : Minimum execution times

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

## ST10X167

### 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). Bit 2 to 6 are not allocated in the ST10F167 device.

F167 Reset Value: 0XX0h

**SYSCON (FF12h / 89h)**

SFR

C/R167 Reset Value: 0X00h

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	XPEN	VISI BLE	XPEN SHARE
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
XPEN-SHARE	<b>XBUS Peripheral Share Mode Control</b> '0': External accesses to XBUS peripherals are disabled '1': XBUS peripherals are accessible via the external bus during hold mode
VISIBLE	<b>Visible Mode Control</b> '0': Accesses to XBUS peripherals are done internally '1': XBUS peripheral accesses are made visible on the external pins
XPEN	<b>XBUS Peripheral Enable Bit</b> For ST10C167 / ST10R167 This Bit is used to enable XRAM and XCAN. For ST10F167 this Bit is used to enable XRAM only as XCAN is always enabled. '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 Not allocated in ST10F167	<b>Bidirectional Reset Enable</b> '0': $\overline{RSTIN}$ pin is an input pin only. SW Reset or WDT Reset have no effect on this pin '1': $\overline{RSTIN}$ pin is a bidirectional pin. This pin is pulled low during 1024 TCL during reset sequence.
OWDDIS Not allocated in ST10F167	<b>Oscillator Watchdog Disable Control</b> '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 $\mu$ s, the CPU clock is switched automatically to PLL's base frequency (around 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 Not allocated in ST10F167	<b>Power Down Mode Configuration Control</b> '0': Power Down Mode can only be entered during PWRDN instruction execution if $\overline{NMI}$ pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must occur by asserting the $\overline{RSTIN}$ pin. '1': Power Down Mode can only be entered during PWRDN instruction execution if all enabled fast external interrupt EXxIN pins are in their inactive level. Exiting this mode can be done by asserting one enabled EXxIN pin.
CSCFG Not allocated in ST10F167	<b>Chip Select Configuration Control</b> '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	<b>Write Configuration Control</b> (Inverted copy of WRC bit of RPOH) '0': Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function '1': Pin $\overline{WR}$ acts as $\overline{WRL}$ , pin $\overline{BHE}$ acts as $\overline{WRH}$
CLKEN	<b>System Clock Output Enable</b> (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	<b>Disable/Enable Control for Pin BHE</b> (Set according to data bus width) '0': Pin $\overline{\text{BHE}}$ enabled '1': Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose I/O
ROMEN	<b>Internal Memory Enable</b> (Set according to pin $\overline{\text{EA}}$ during reset) '0': Internal ROM disabled: accesses to the Flash Memory area use the external bus '1': Internal ROM enabled
SGTDIS	<b>Segmentation Disable/Enable Control</b> '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit) '1': Segmentation disabled (Only IP is saved/restored)
ROMS1	<b>Internal Memory Mapping</b> '0': Internal ROM area mapped to segment 0 (00'0000h...00'7FFFh) '1': Internal ROM area mapped to segment 1 (01'0000h...01'7FFFh)
STKSZ	<b>System Stack Size</b> 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 8.4 - 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 ( $f_{\text{OUT}} = f_{\text{CPU}}$ ).

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 64K Byte (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 Bitfield defines the size of the physical system stack, which is located in the internal RAM of the ST10X167. 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 Bitfield STKSZ are described in more detail in Stack Operations (see Section 21.1 - Stack Operations).

#### 4.4.2 - 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	<b>Negative Result</b> - Set, when the result of an ALU operation is negative.
C	<b>Carry Flag</b> - Set, when the result of an ALU operation produces a carry Bit.
V	<b>Overflow Result</b> - Set, when the result of an ALU operation produces an overflow.
Z	<b>Zero Flag</b> - Set, when the result of an ALU operation is zero.
E	<b>End of Table Flag</b> - Set, when the source operand of an instruction is 8000h or 80h.
MULIP	<b>Multiplication/Division In Progress</b> '0': There is no multiplication/division in progress. '1': A multiplication/division has been interrupted.
USR0	<b>User General Purpose Flag</b> - May be used by the application software.
HLDEN, ILVL, IEN	<b>Interrupt and EBC Control Fields</b> Define the response to interrupt requests and enable external bus Arbitration. (Described in Chapter 5 - 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	$0 < \text{Rounding error} < \frac{1}{2} \text{ LSB}$
1	0	$\text{Rounding error} = \frac{1}{2} \text{ LSB}$
1	1	$\text{Rounding error} > \frac{1}{2} \text{ LSB}$

**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 5 - Interrupt and Trap Functions.

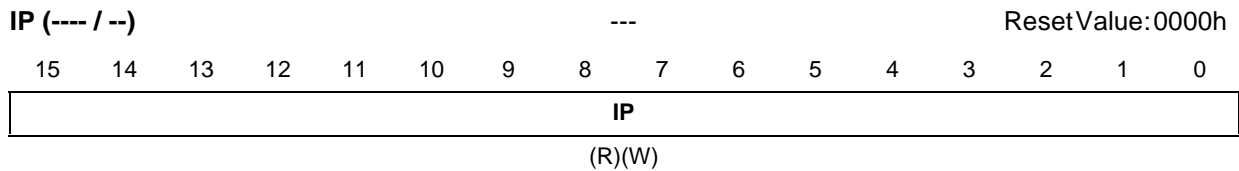
After reset all interrupts are globally disabled, and the lowest priority (ILVL=0) is assigned to the initial CPU activity.

**4.4.3 - 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.

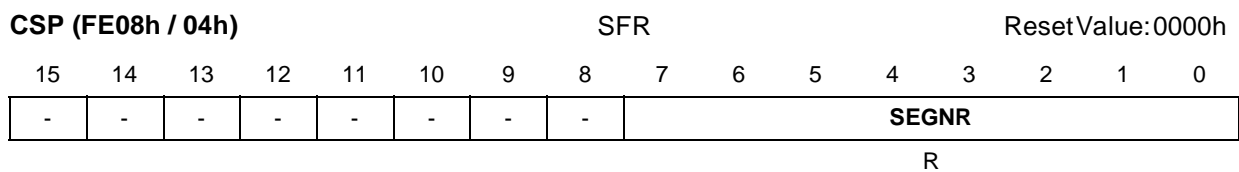
The IP register is implicitly updated by the CPU for branch instructions and after instruction fetch operations.



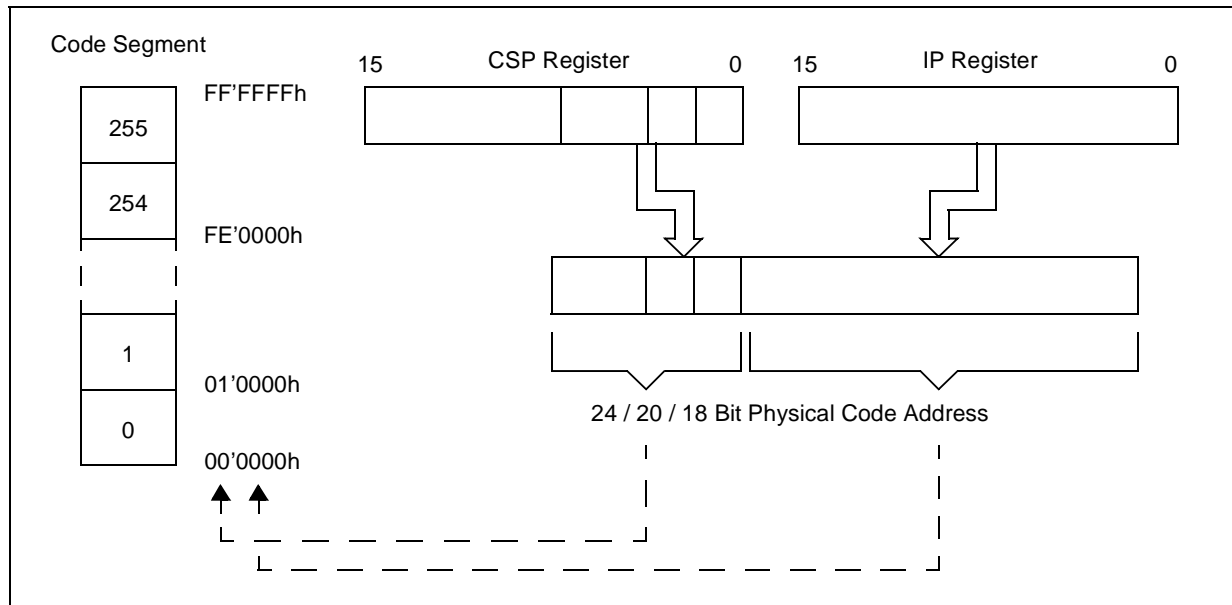
Bit	Function
IP	Specifies the intra segment offset, from where the current instruction is to be fetched. IP refers to the current segment (SEGNR).

**4.4.4 - 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 Bit of register CSP select one of up to 256 segments of 64K Byte each, while the upper 8 Bit are reserved for future use.



Bit	Function
SEGNR	<b>Segment Number:</b> 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.

**4.4.5 - 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 Bit of each DPP register select one of the 1024 possible 16K Byte data pages while the upper 6 Bit are reserved for future use. The DPP registers make it possible to access the entire memory space, in pages of 16K Byte 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	<b>Data Page Number of DPPx:</b> Specifies the data page selected via DPPx. Only the 2 least significant Bit of DPPx are used when segmentation is disabled.

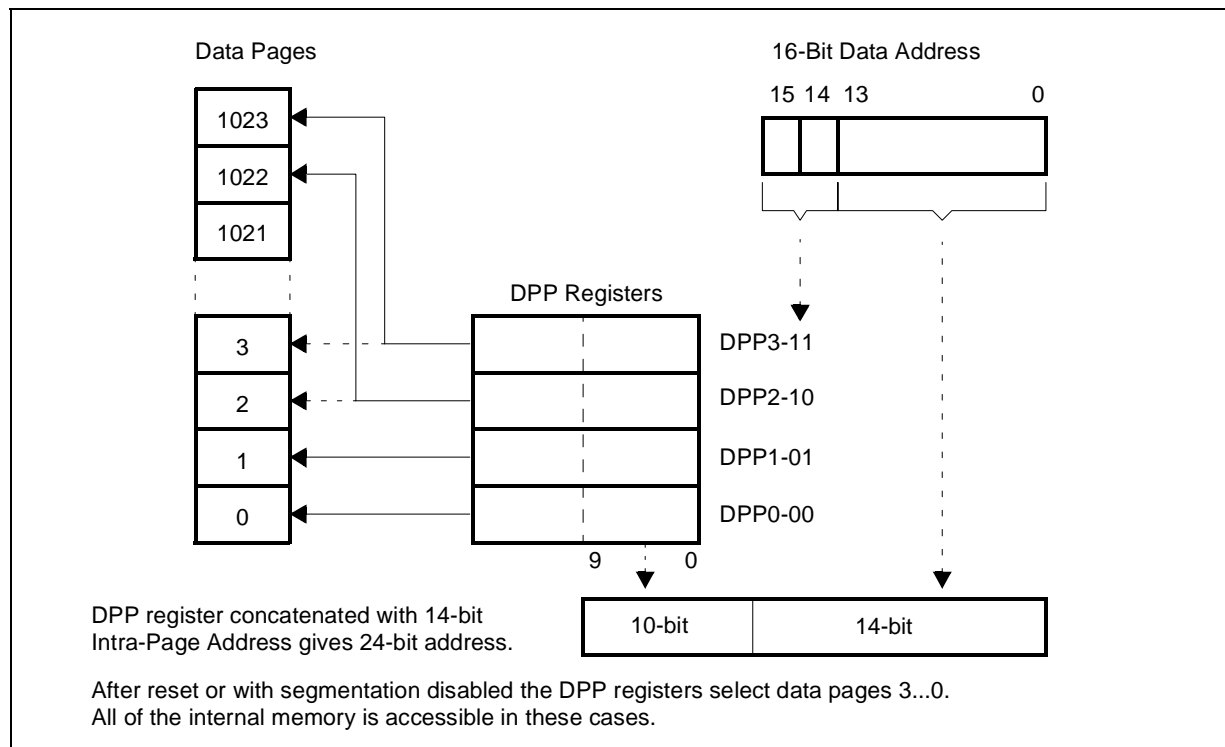
Data paging is performed by concatenating the lower 14 Bit of an indirect or direct long 16 Bit address with the contents of the DDP register selected by the upper two Bit of the 16 Bit address. The content of the selected DPP register specifies one of the 1024 possible data pages. This data page base 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



#### 4.4.6 - The Context Pointer CP

This non Bit addressable register is used to select the current register context. This means that the CP register value determines the address of the first General Purpose Register (GPR) within the current register bank of up to 16 Wordwide and/or 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	<b>Modifiable portion of register CP:</b> 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 (2K Byte).
- 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

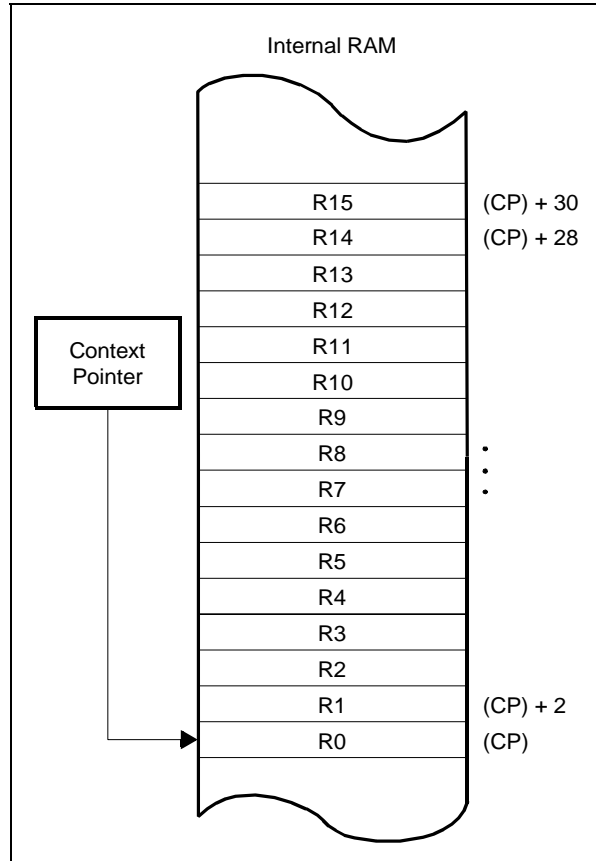
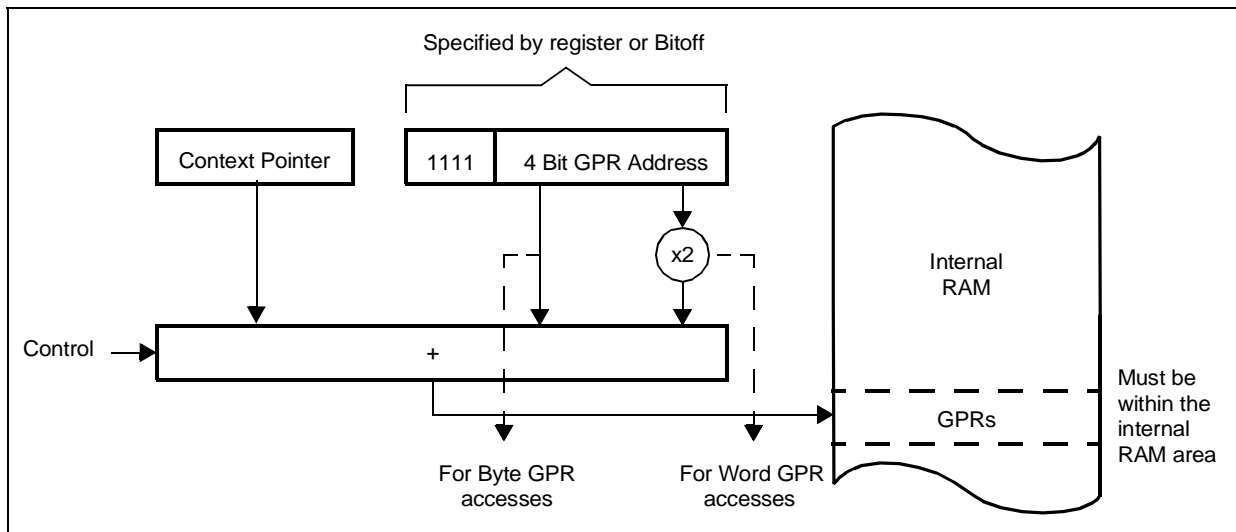


Figure 17 : Implicit CP use by short GPR addressing modes



#### 4.4.7 - 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 ResetValue: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	<b>Modifiable portion of register SP:</b> Specifies the top of the internal system stack.

#### 4.4.8 - 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 ResetValue: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	<b>Modifiable portion of register STKOV:</b> 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 21 - System Programming.

**4.4.9 - 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.

<b>STKUN (FE16h / 0Bh)</b>				<b>SFR</b>				<b>ResetValue:FC00h</b>							
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	<b>Modifiable portion of register STKUN:</b> 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 21 - 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.

**4.4.10 - 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 Bit of the 32 Bit dividend before the division is started. After any division, register MDH represents the 16 Bit remainder.

<b>MDH (FE0Ch / 06h)</b>				<b>SFR</b>				<b>ResetValue:0000h</b>							
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 Bit 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 21 - System Programming.

#### 4.4.11 - 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 Bit of the 32 Bit result. For long divisions, the MDL register must be loaded with the low order 16 Bit 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 Bit 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 21 - System Programming.

#### 4.4.12 - 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	<b>Multiply/Divide Register In Use</b> '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	<b>Internal Machine Status</b> 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 21 - System Programming.

#### 4.4.13 - The Constant Zeros Register ZEROS

All Bit 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

#### 4.4.14 - The Constant Ones Register ONES

All Bit 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

#### 4.4.15 - 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 - INTERRUPT AND TRAP FUNCTIONS

The architecture of the ST10X167 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 ST10X167'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 ST10X167 does not provide dedicated interrupt pins, it allows to connect external interrupt sources and provides several mechanisms to react on external events, including standard inputs, non-maskable interrupts and fast

external interrupts. These interrupt functions are alternate port functions, except for the non-maskable interrupt and the reset input.

### 5.1 - Interrupt System Structure

The ST10X167 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 ST10X167, 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 ST10X167 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 ST10X167'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 6 lists all sources that are capable of requesting interrupt or PEC service in the ST10X167, 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 Byte per entry).

**Table 6 :** 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



**Table 6** : 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 26	CC26IR	CC26IE	CC26INT	00'00E8h	3Ah
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
CAN Interface <sup>1</sup>	XP0IR	XP0IE	XP0INT	00'0100h	40h
X-Peripheral node 1 <sup>2</sup>	XP1IR	XP1IE	XP1INT	00'0104h	41h
X-Peripheral node 2 <sup>2</sup>	XP2IR	XP2IE	XP2INT	00'0108h	42h
PLL Unlock	XP3IR	XP3IE	XP3INT	00'010Ch	43h

Notes 1. For devices which do not incorporate a CAN module or a PLL, the respective interrupt nodes may be used for software triggered interrupts (see Section 8.7 - The XBUS Interface).

2. The currently unused nodes in the table (X-Peripheral nodes) are prepared to accept interrupt requests from integrated XBUS peripherals. Nodes, where no X-Peripherals are connected or when no PLL is implemented, may be used to generate software controlled interrupt requests by setting the respective XPnIR bit.

**Table 7 :** Vector locations and status for hardware traps

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

The Table 7 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.

**5.1.1 - 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.

### 5.1.2 - 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.

### 5.1.3 - Interrupt Control Registers

All interrupt control registers are identically organized. The lower 8 bit 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 bit of the respective register are reserved. All interrupt control registers are bit-addressable and all bit 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 bit (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 Area				ResetValue: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	<b>Group Level</b> Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
ILVL	<b>Interrupt Priority Level</b> Defines the priority level for the arbitration of requests. Fh: Highest priority level 0h: Lowest priority level
xxIE	<b>Interrupt Enable Control Bit</b> (individually enables/disables a specific source) '0': Interrupt Request is disabled '1': Interrupt Request is enabled
xxIR	<b>Interrupt Request Flag</b> '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.

**5.1.4 - Interrupt Priority Level and Group Level**

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

When more than one interrupt request on a specific level 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 bit field ILVL of register PSW after pushing the old PSW contents on the stack.

The interrupt system of the ST10X167 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.

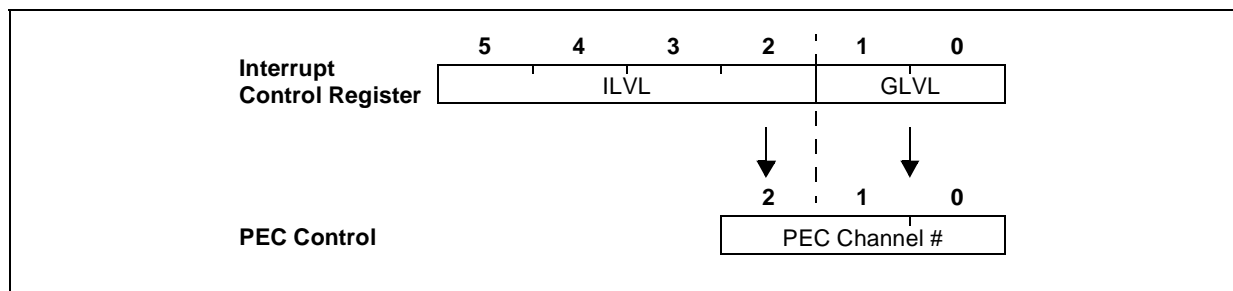
**Note** Priority level 0000b is the default level of the CPU. Therefore a request on level 0 will never be serviced, because it can never interrupt the CPU. However, an enabled interrupt request on level 0000b will terminate the ST10X167's Idle mode and reactivate the CPU.

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 18). 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 18).

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 18** : 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.

### 5.1.5 - 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 ST10X167 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	RW

Bit	Function
N, C, V, Z, E, MULIP, USR0	<b>CPU status flags</b> (Described in section "The Central Processing Unit") Define the current status of the CPU (ALU, multiplication unit).
HLDEN	<b>HOLD Enable</b> (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	<b>CPU Priority Level</b> Defines the current priority level for the CPU Fh: Highest priority level 0h: Lowest priority level
IEN	<b>Interrupt Enable Control Bit</b> (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.

**5.2 - 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.

<b>PECCx (FECyh / 6zh, see Table 8)</b>						<b>SFR</b>		<b>ResetValue:0000h</b>							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	<b>INC</b>		<b>BWT</b>	<b>COUNT</b>							
RW						RW		RW							

Bit	Function
COUNT	<b>PEC Transfer Count</b> Counts PEC transfers and influences the channel's action (see table below)
BWT	<b>Byte / Word Transfer Selection</b> 0: Transfer a Word 1: Transfer a Byte
INC	<b>Increment Control</b> (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 8 : 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')	<b>No action!</b> 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 ST10X167 just below the bit-addressable area (see Figure 19).

**Figure 19 :** Mapping of PEC pointers into the internal RAM

<b>DSTP7</b>	00'FCFEh	<b>DSTP3</b>	00'FCEEh
<b>SRCP7</b>	00'FCFCh	<b>SRCP3</b>	00'FCECh
<b>DSTP6</b>	00'FCFAh	<b>DSTP2</b>	00'FCEAh
<b>SRCP6</b>	00'FCF8h	<b>SRCP2</b>	00'FCE8h
<b>DSTP5</b>	00'FCF6h	<b>DSTP1</b>	00'FCE6h
<b>SRCP5</b>	00'FCF4h	<b>SRCP1</b>	00'FCE4h
<b>DSTP4</b>	00'FCF2h	<b>DSTP0</b>	00'FCE2h
<b>SRCP4</b>	00'FCF0h	<b>SRCP0</b>	00'FCE0h

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.

### **5.3 - 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.

#### **5.3.1 - 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, discarding 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 21 - System Programming).

#### **5.3.2 - 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 ST10X167 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 9** : 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!

### 5.4 - Saving the Status During Interrupt Service

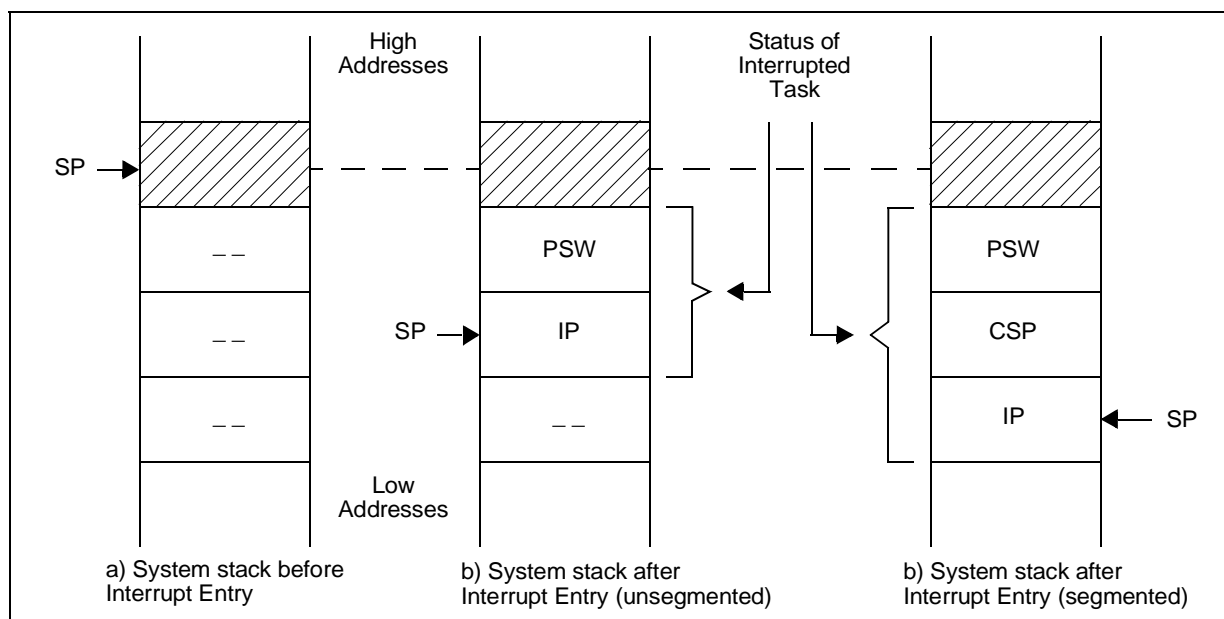
Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved along with the location, where the execution of the interrupted task is to be resumed after returning from the service routine.

This return location is specified through the Instruction Pointer (IP) and, in case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register SYSCON 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 20** : 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.

**5.4.1 - 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 ST10X167 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).

**5.5 - Interrupt Response Times**

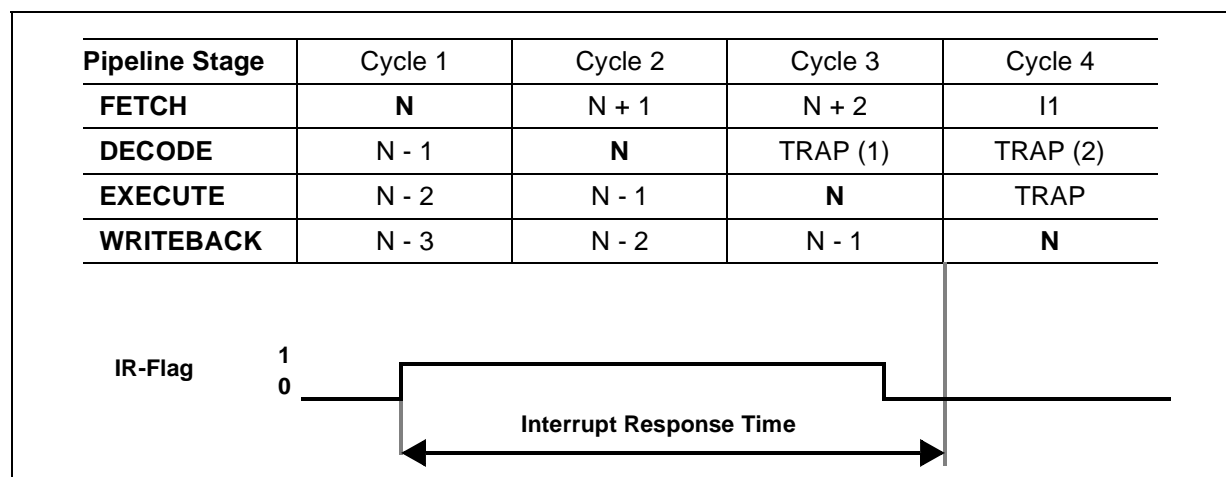
The interrupt response time defines the time from an interrupt request flag of an enabled interrupt source being set until the first instruction (I1) being fetched from the interrupt vector location. The basic interrupt response time for the ST10X167 is 3 instruction cycles (see Figure 21).

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 (waitstates) therefore influences the interrupt response time.

In the Figure 21 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 21** : Pipeline diagram for interrupt response time



The minimum interrupt response time is 5 CPU clock cycles. This requires program execution from the internal ROM, 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 ROM 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 ROM 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 ROM, the interrupt response time is 7 Word bus accesses plus 2 CPU clock cycles, because fetching of instruction I1 from internal ROM 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 ROM, 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.

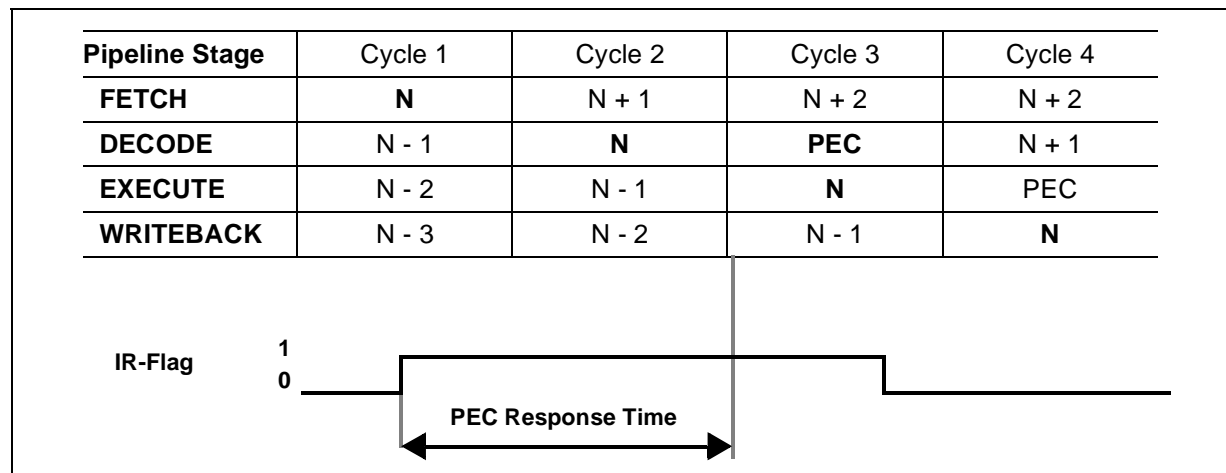
In most cases two instructions will be executed during this time. Only one instruction will typically be executed, if the first instruction following the RETI instruction is a branch instruction (without cache hit), or if it reads an operand from internal ROM, or if it is executed out of the internal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

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

### 5.5.1 - 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 ST10X167 is 2 instruction cycles.

Figure 22 : Pipeline diagram for PEC response time



In the Figure 22 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 ROM, 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 ROM, 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 ROM 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 ROM 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 ROM 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 READY signal or by bus arbitration (HOLD mode).

## 5.6 - External Interrupts

Although the ST10X167 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 10).

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 10).

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 10** : 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  $\overline{NMI}$  and the reset input  $\overline{RSTIN}$  provide another possibility for the CPU to react on an external input signal.  $\overline{NMI}$  and  $\overline{RSTIN}$  are dedicated input pins, which cause hardware traps.

**5.6.1 - 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 ST10X167 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.

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 detailed in Chapter 19 - Power Reduction Modes.

EXICON (F1C0h / E0h)								ESFR								ResetValue: 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)	<p><b>External Interrupt x Edge Selection Field (x=3...0)</b></p> <p>0 0: Fast external interrupts disabled: standard mode EXxIN pin not taken in account for entering/exiting Power Down mode.</p> <p>0 1: Interrupt on positive edge (rising) Enter Power Down mode if EXiIN = '0', exit if EXxIN = '1' (ref as 'high' active level)</p> <p>1 0: Interrupt on negative edge (falling) Enter Power Down mode if EXiIN = '1', exit if EXxIN = '0' (ref as 'low' active level)</p> <p>1 1: Interrupt on any edge (rising or falling) Always enter Power Down mode, exit if EXxIN level changed.</p>



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  $\neq$  00b). However, general purpose I/O is possible in all cases.

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

## 5.7 - 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 ST10X167 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.

### 5.7.1 - 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.

### 5.7.2 - 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 ST10X167 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 5.1 - 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 ST10X167 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 (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
  - 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'.

## ST10X167

TFR (FFACh / D6h)						SFR						Reset Value: 0000h			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NMI	STK OF	STK UF	-	-	-	-	-	UND OPC	-	-	-	PRT FLT	ILL OPA	ILL INA	ILL BUS
RW	RW	RW						RW				RW	RW	RW	RW

Bit	Function
ILLBUS	<b>Illegal External Bus Access Flag</b> An external access has been attempted with no external bus defined.
ILLINA	<b>Illegal Instruction Access Flag</b> A branch to an odd address has been attempted.
ILLOPA	<b>Illegal Word Operand Access Flag</b> A Word operand access (read or write) to an odd address has been attempted.
PRTFLT	<b>Protection Fault Flag</b> A protected instruction with an illegal format has been detected.
UNDOPC	<b>Undefined Opcode Flag</b> The currently decoded instruction has no valid ST10X167 opcode.
STKUF	<b>Stack Underflow Flag</b> The current stack pointer value exceeds the content of register STKUN.
STKOF	<b>Stack Overflow Flag</b> The current stack pointer value falls below the content of register STKOV.
NMI	<b>Non Maskable Interrupt Flag</b> A negative transition (falling edge) has been detected on pin $\overline{\text{NMI}}$ .

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

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

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

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



### 5.7.3 - External NMI Trap

Whenever a high to low transition on the dedicated external 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.

### 5.7.4 - 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.

### 5.7.5 - 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.

### 5.7.6 - Undefined Opcode Trap

When the instruction currently decoded by the CPU does not contain a valid ST10X167 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.

### 5.7.7 - 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.

### 5.7.8 - 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.

### 5.7.9 - 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.

### 5.7.10 - Illegal External Bus Access Trap

Whenever the CPU requests an external instruction fetch, data read or data write, and no external bus configuration has been specified, the ILLBUS flag in register TFR is set and the CPU enters the illegal bus access trap routine.

The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

## 6 - PARALLEL PORTS

### 6.1 - Introduction

The ST10X167 has up to 111 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),
- One 16 Bit input port (Port5),
- One 16 Bit I/O port (Port2).

These port lines may be used for general purpose Input/Output, controlled via software, or may be used implicitly by ST10X167'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). The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs.

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

#### 6.1.1 - Open Drain Mode

Some of I/O Ports of ST10X167 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 and P8 (see respective sections), and is controlled through the respective Open Drain Control Registers ODPx.

These registers allow the individual Bitwise 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 24).

Figure 23 : SFRs and pins associated with the parallel ports

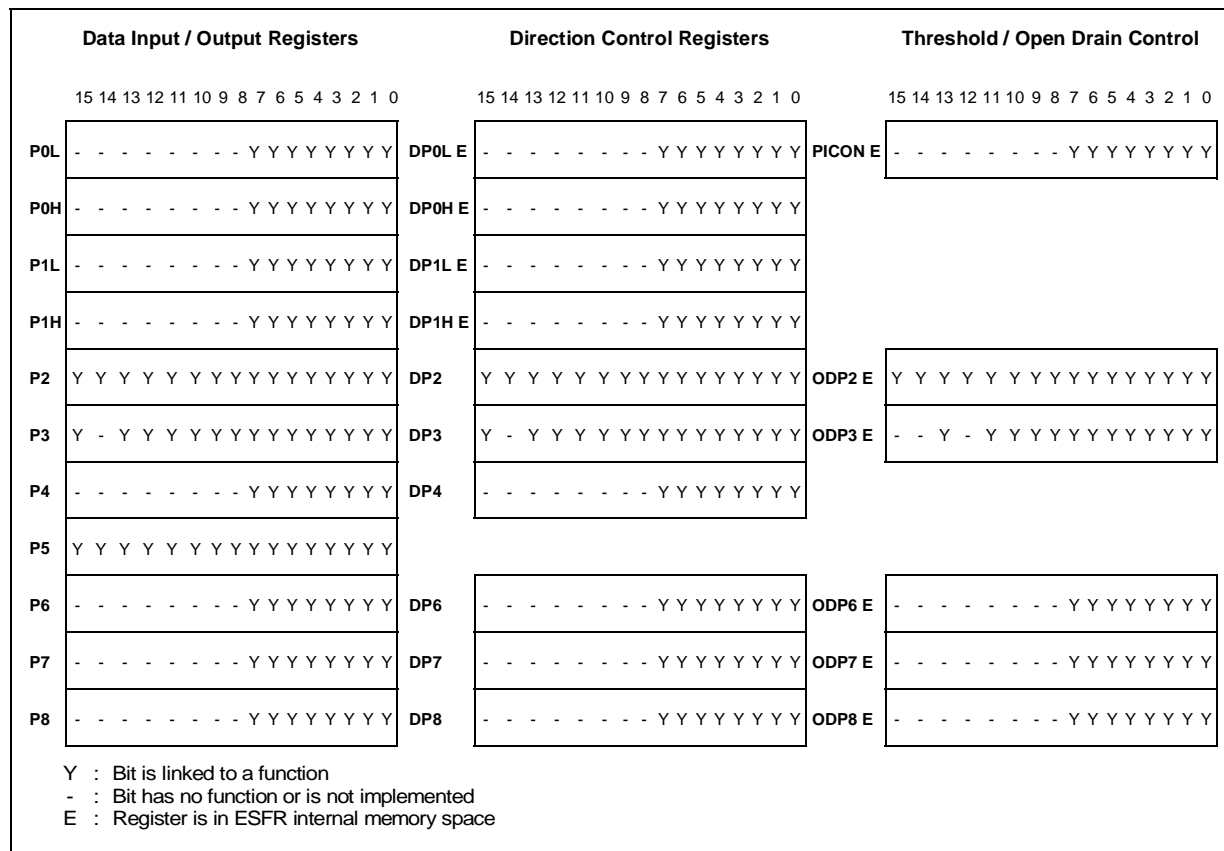
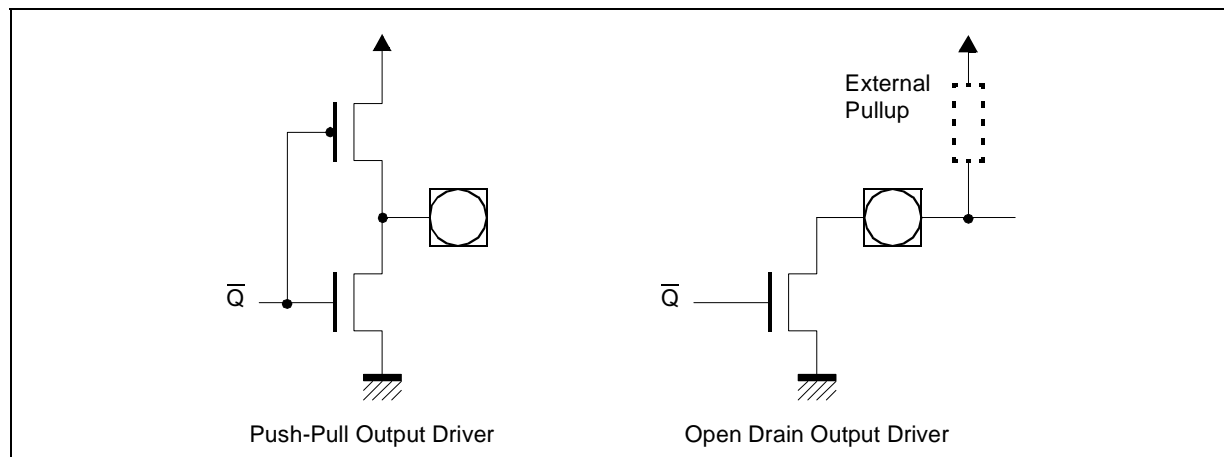


Figure 24 : Output drivers in push-pull mode and in open drain mode



6.1.2 - Input Threshold Control

The standard inputs of the ST10X167 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, 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.

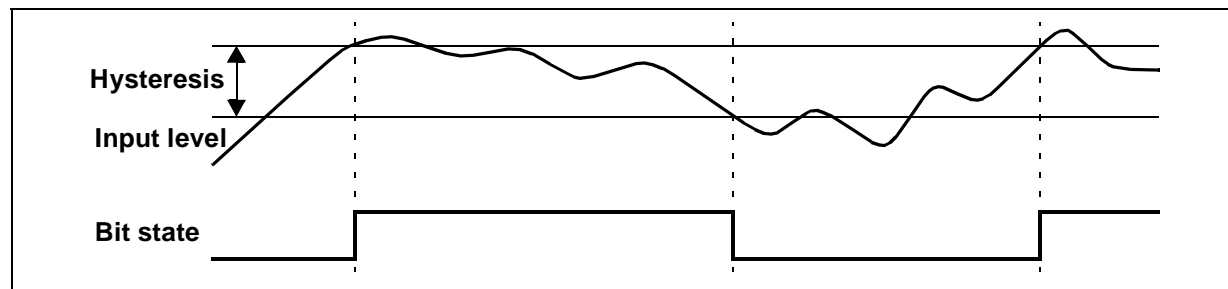
The Port Input Control register PICON is used to select these thresholds for each Byte of the indicated ports, the 8 Bit ports P7 and P8 are controlled by one Bit each while ports P2 and P3 are controlled by two Bit each.

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	-	-	P3HIN	P3LIN	P2HIN	P2LIN
								RW	RW			RW	RW	RW	RW

Bit	Function
PxLIN	<b>Port x Low Byte Input Level Selection</b> 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	<b>Port x High Byte Input Level Selection</b> 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 25 : Hysteresis for special input thresholds



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

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

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

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

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

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

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

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

```
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
```

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").

## 6.2 - 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 DPOH and DPOL.

### 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	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	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	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	DP0L.7	DP0L.6	DP0L.5	DP0L.4	DP0L.3	DP0L.2	DP0L.1	DP0L.0	DP0L.7	DP0L.6	DP0L.5	DP0L.4	DP0L.3	DP0L.2	DP0L.1	DP0L.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	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	DP0H.7	DP0H.6	DP0H.5	DP0H.4	DP0H.3	DP0H.2	DP0H.1	DP0H.0	DP0H.7	DP0H.6	DP0H.5	DP0H.4	DP0H.3	DP0H.2	DP0H.1	DP0H.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	<b>Port direction register DP0H or DP0L Bit y</b> DP0X.y = 0: Port line P0X.y is an input (high-impedance) DP0X.y = 1: Port line P0X.y is an output

**6.2.1 - 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.

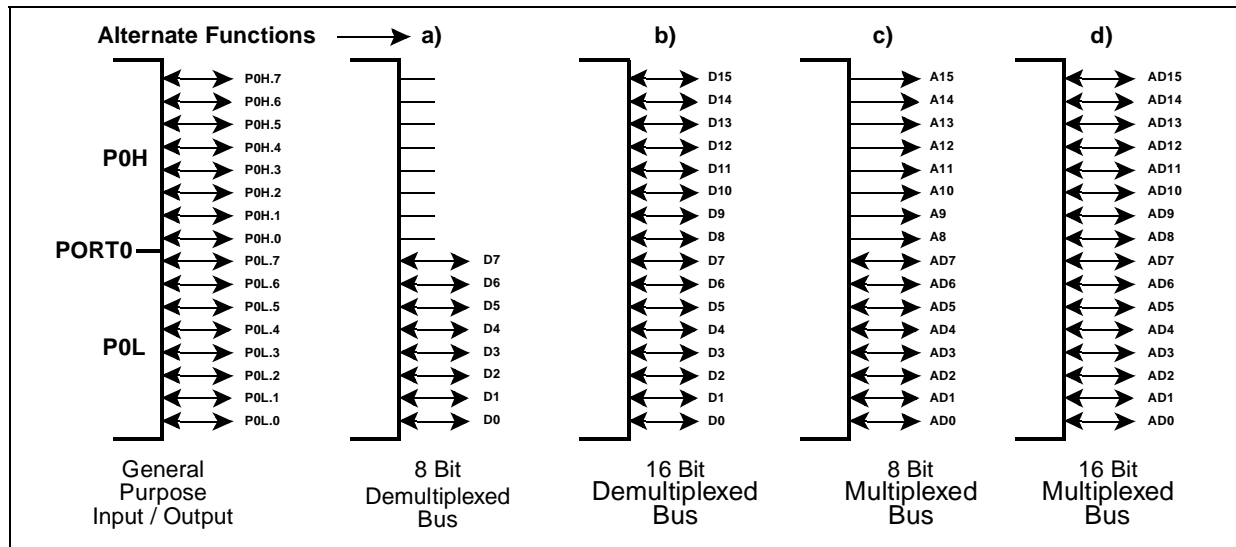
During external accesses in multiplexed bus modes PORT0 first outputs the 16 Bit intra-segment address as an alternate output function. PORT0 is then switched to high-impedance input mode to read the incoming instruction or data.

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 26).

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.

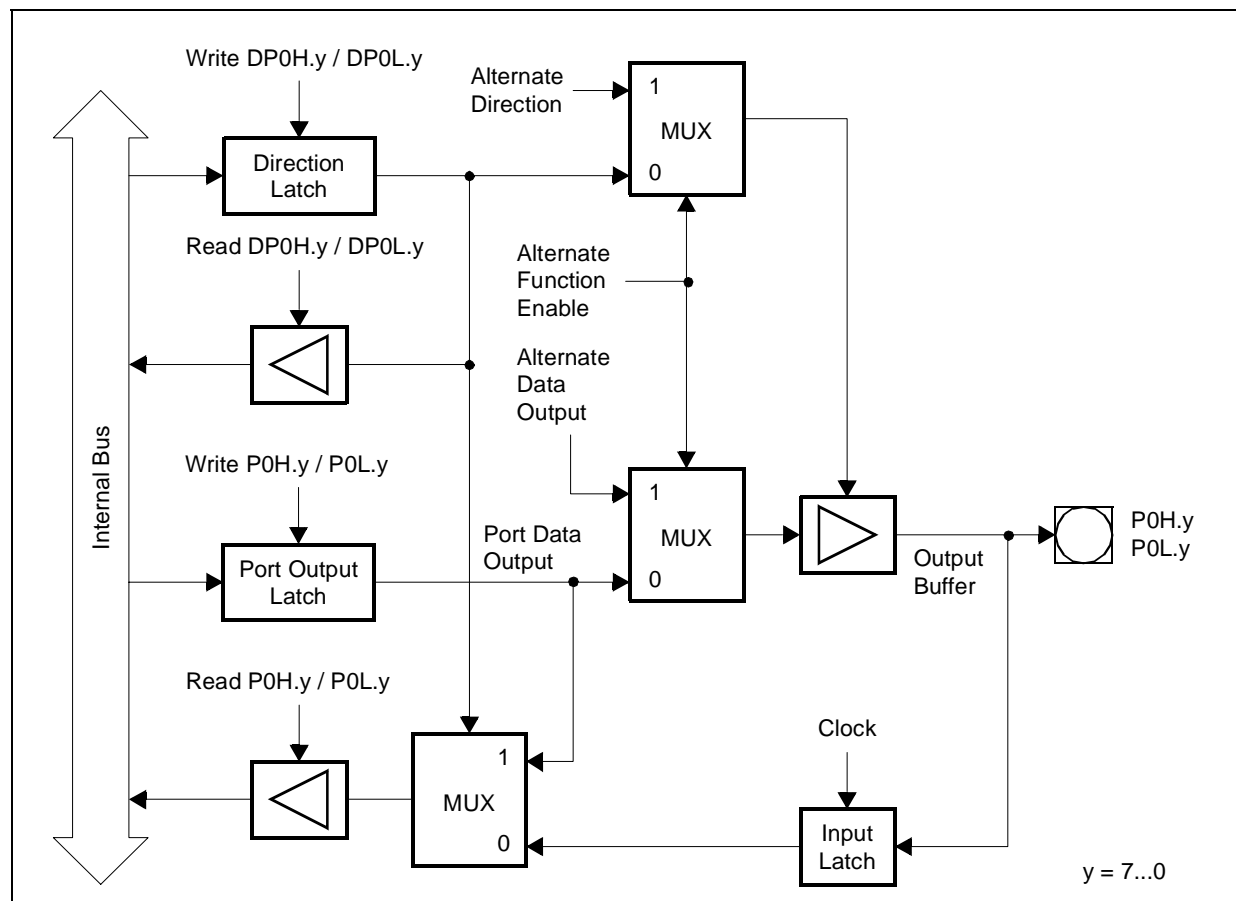


Figure 26 : PORT0 I/O and alternate functions



The Figure 27 shows the structure of a PORT0 pin.

Figure 27 : Block diagram of a PORT0 pin



**6.3 - 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	<b>Port direction register DP1H or DP1L Bit y</b> DP1X.y = 0: Port line P1X.y is an input (high-impedance) DP1X.y = 1: Port line P1X.y is an output

**6.3.1 - 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.

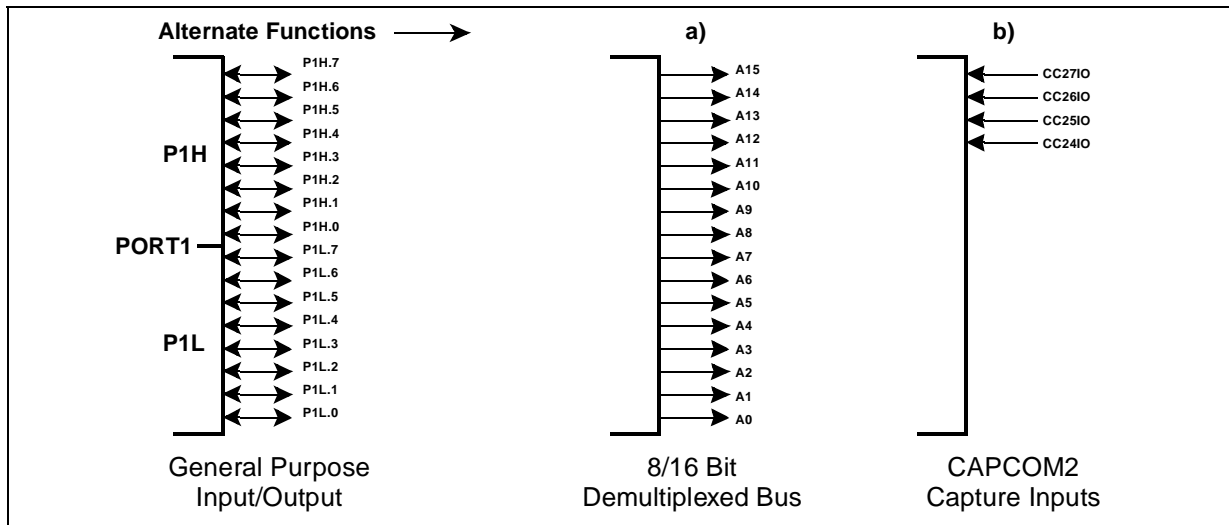
As a side effect, the capture input capability of these lines can also be used in the address bus mode. Hereby changes of the upper address lines could be detected and trigger an interrupt request in order to perform some special service routines. External capture signals can only be applied if no address 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 28 : PORT1 I/O and alternate functions

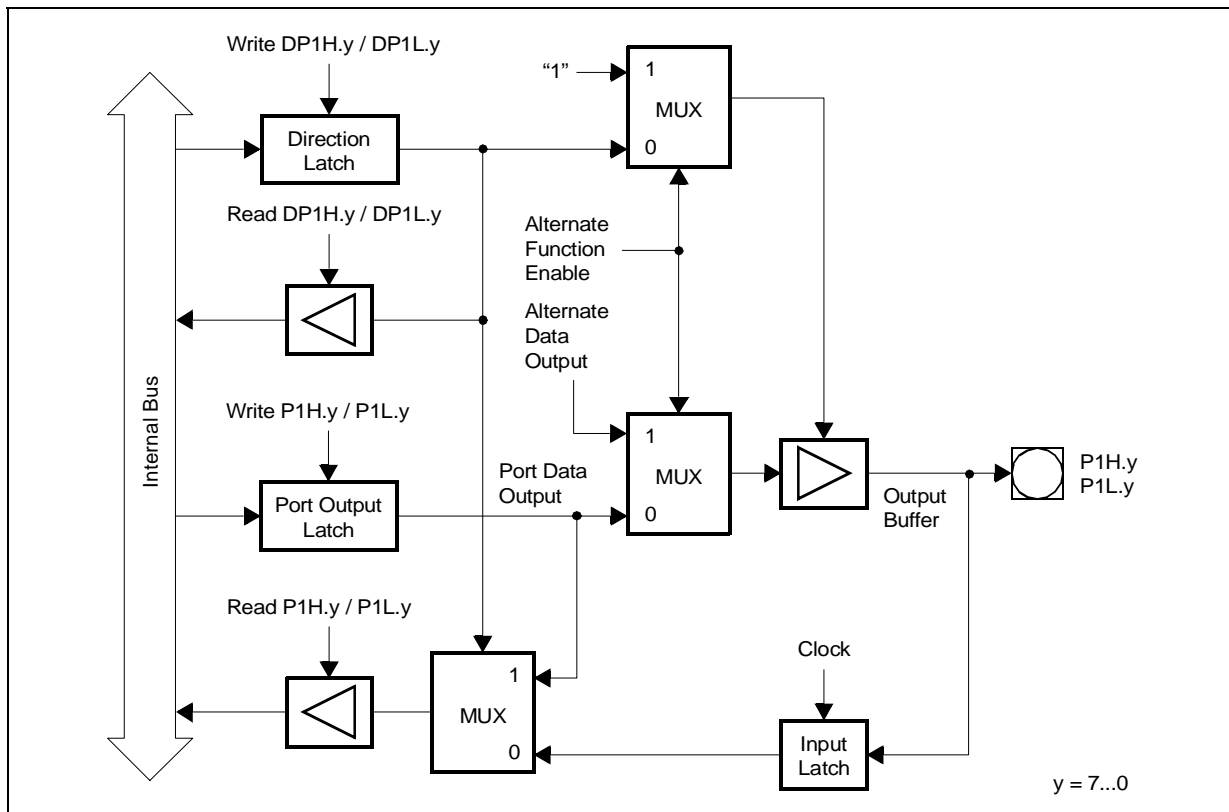


When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled “Alternate Data Output” via 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 29 shows the structure of a PORT1 pin.

Figure 29 : Block diagram of a PORT1 pin



6.4 - 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
<b>P2.15</b>	<b>P2.14</b>	<b>P2.13</b>	<b>P2.12</b>	<b>P2.11</b>	<b>P2.10</b>	<b>P2.9</b>	<b>P2.8</b>	<b>P2.7</b>	<b>P2.6</b>	<b>P2.5</b>	<b>P2.4</b>	<b>P2.3</b>	<b>P2.2</b>	<b>P2.1</b>	<b>P2.0</b>
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
P2.y	<b>Port data register P2 Bit y</b>

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

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

Bit	Function
DP2.y	<b>Port direction register DP2 Bit y</b> 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
<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>	<b>ODP2</b>
.15	.14	.13	.12	.11	.10	.9	.8	.7	.6	.5	.4	.3	.2	.1	.0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
ODP2.y	<b>Port2 Open Drain control register Bit y</b> 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

### 6.4.1 - 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 31), 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 P2.15...P2.0 can also be used as external interrupt inputs with a sample rate of 8 CPU clock cycles.

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

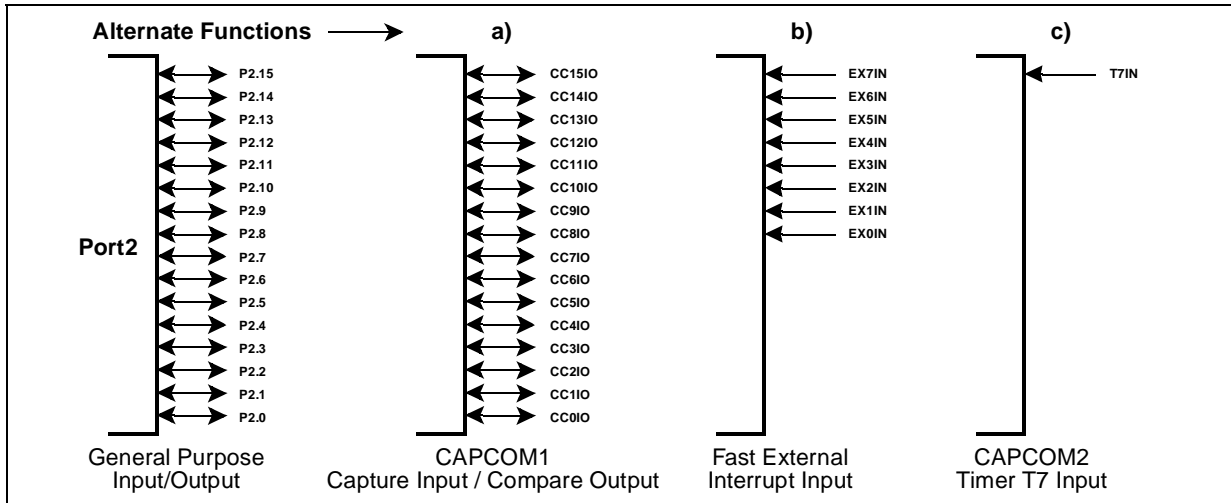
P2.15 in addition is the input for CAPCOM2 timer T7 (T7IN).

The Table 11 summarizes the alternate functions of Port2.

**Table 11** : Port2 alternate functions

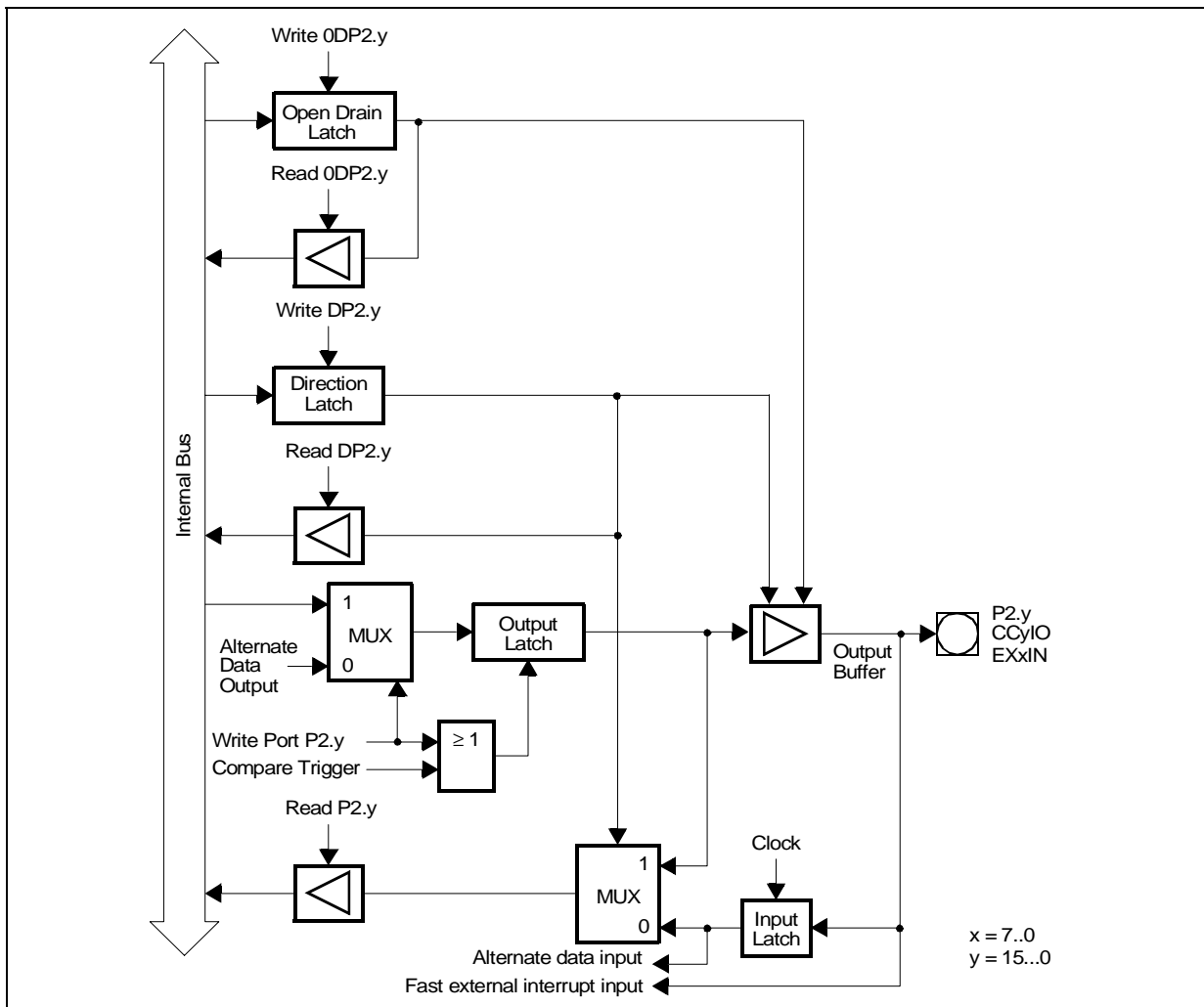
P.2 Pin	Alt Function a)	Alternate Function b)	Alternate Function c)
P2.0	CC0IO	-	-
P2.1	CC1IO	-	-
P2.2	CC2IO	-	-
P2.3	CC3IO	-	-
P2.4	CC4IO	-	-
P2.5	CC5IO	-	-
P2.6	CC6IO	-	-
P2.7	CC7IO	-	-
P2.8	CC8IO	EX0IN Fast External Interrupt 0 Input	-
P2.9	CC9IO	EX1IN Fast External Interrupt 1 Input	-
P2.10	CC10IO	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 30 : 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 31 : Block diagram of a Port2 pin



**6.5 - 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

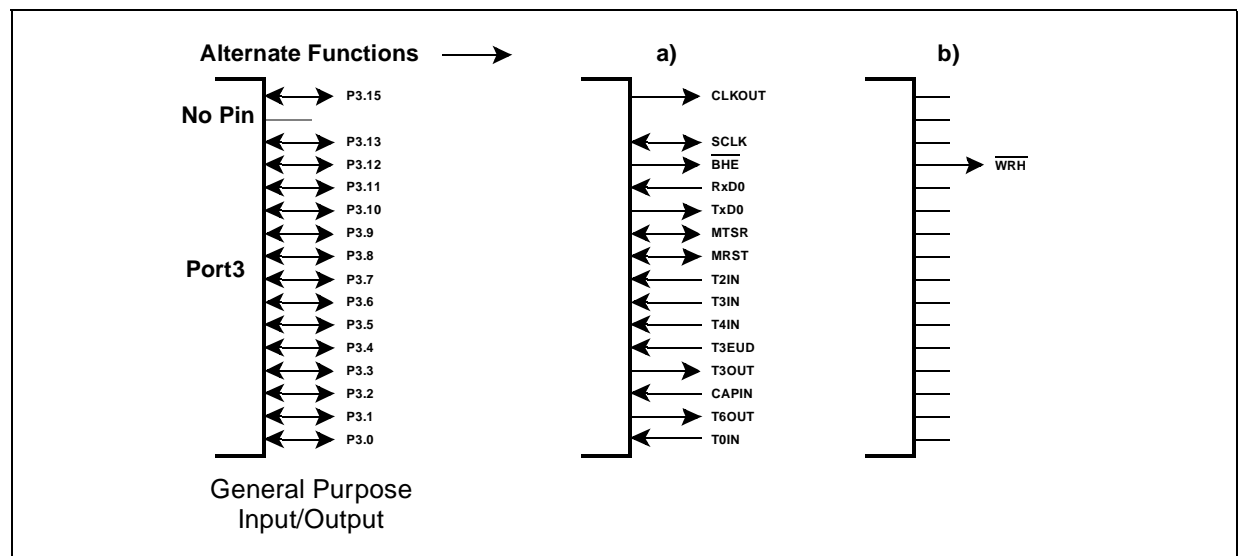
6.5.1 - Alternate Functions of Port3

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 12 : 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
P3.12	$\overline{\text{BHE}}/\text{WRH}$	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 32 : Port3 I/O and alternate functions



The port structure of the Port3 pins depends on their alternate function (see Figure 33).

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.

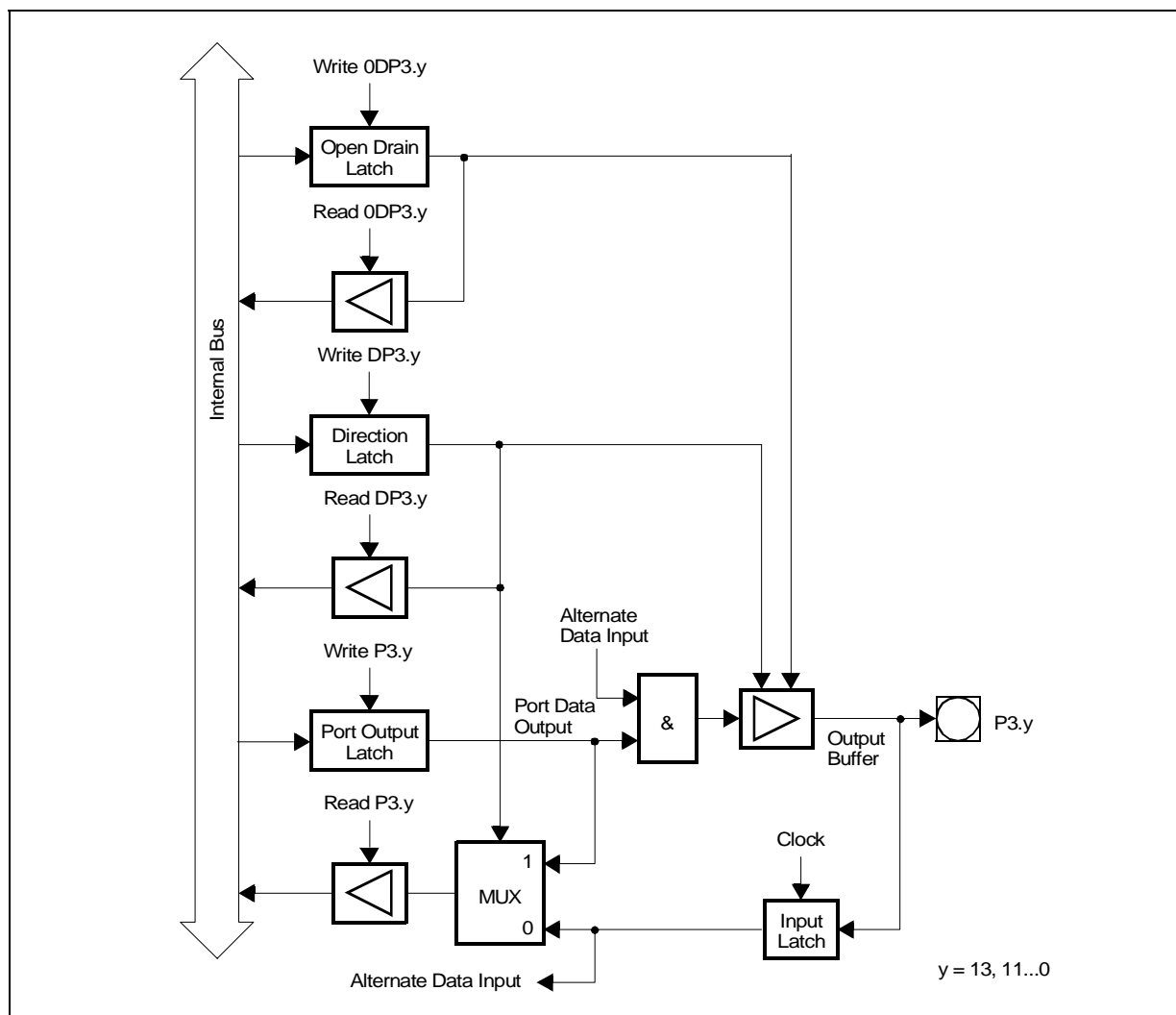
When the on-chip peripheral associated with a Port3 pin is configured to use the alternate output function, its “Alternate Data Output” line is ANDed with the port output latch line. When using these alternate functions, the user must set the direction of the port line to output (DP3.y=1) and must set 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 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 33** : Block diagram of Port3 pin with alternate input or alternate output function

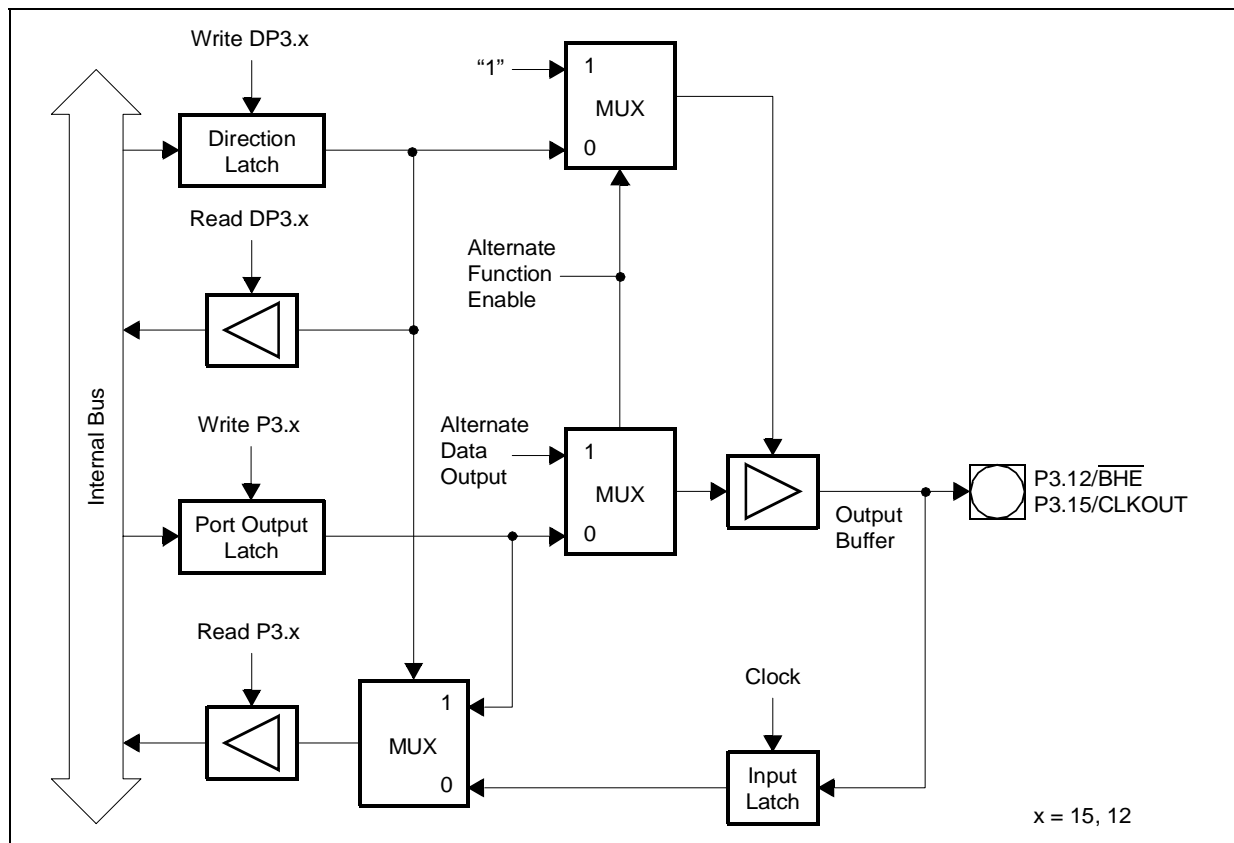


Pin P3.12 ( $\overline{\text{BHE}}/\overline{\text{WRH}}$ ) is another pin with an alternate output function, however, its structure is slightly different (see Figure 34). After reset the  $\overline{\text{BHE}}$  or  $\overline{\text{WRH}}$  function must be used depending on the system start-up configuration. In either of these cases, there is no possibility to program any port latches before. Thus, the appropriate alternate function is selected automatically. If  $\overline{\text{BHE}}/\overline{\text{WRH}}$  is not used in the system, this pin can be

used for general purpose I/O by disabling the alternate function ( $\text{BYTDIS} = '1' / \text{WRCFG} = '0'$ ).

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

**Figure 34 :** Block diagram of pins P3.15 ( $\overline{\text{CLKOUT}}$ ) and P3.12 ( $\overline{\text{BHE}}/\overline{\text{WRH}}$ )



**6.6 - 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



<b>DP4 (FFCAh / E5h)</b>								<b>SFR</b>								<b>Reset Value: -- 00h</b>															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	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	-	-	-	-	-	-	-	-	DP4.7	DP4.6	DP4.5	DP4.4	DP4.3	DP4.2	DP4.1	DP4.0
								RW																							

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

**6.6.1 - Alternate Functions of Port4**

During external bus cycles that use segmentation (for address space above 64K Byte) 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 Bitfield SALSEL in register RP0H (read only) in order to check the configuration during run time.

Devices with a CAN interface use 2 pins of Port4 to interface the CAN module to an external CAN transceiver. In this case the number of possible segment address lines is reduced.

The Table 13 summarizes the alternate functions of Port4 depending on the number of selected segment address lines (coded via Bitfield SALSEL).

**Table 13 : Port4 Alternate Functions**

P.4 Pin	Standard Function SALSEL = 01 64K Byte	Alternate. Function SALSEL = 11 256K Byte	Alternate Function SALSEL = 00 1M Byte	Alternate Function SALSEL = 10 16M Byte
P4.0	General purpose I/O	Segment address A16	Segment address A16	Segment address A16
P4.1	General purpose I/O	Segment address A17	Segment address A17	Segment address A17
P4.2	General purpose I/O	General purpose I/O	Segment address A18	Segment address A18
P4.3	General purpose I/O	General purpose I/O	Segment address A19	Segment address A19
P4.4	General purpose I/O	General purpose I/O	General purpose I/O	Segment address A20
P4.5	General purpose I/O	General purpose I/O	General purpose I/O	Segment address A21
P4.6	General purpose I/O	General purpose I/O	General purpose I/O	Segment address A22
P4.7	General purpose I/O	General purpose I/O	General purpose I/O	Segment address A23

**Figure 35 : Port4 I/O and alternate functions**

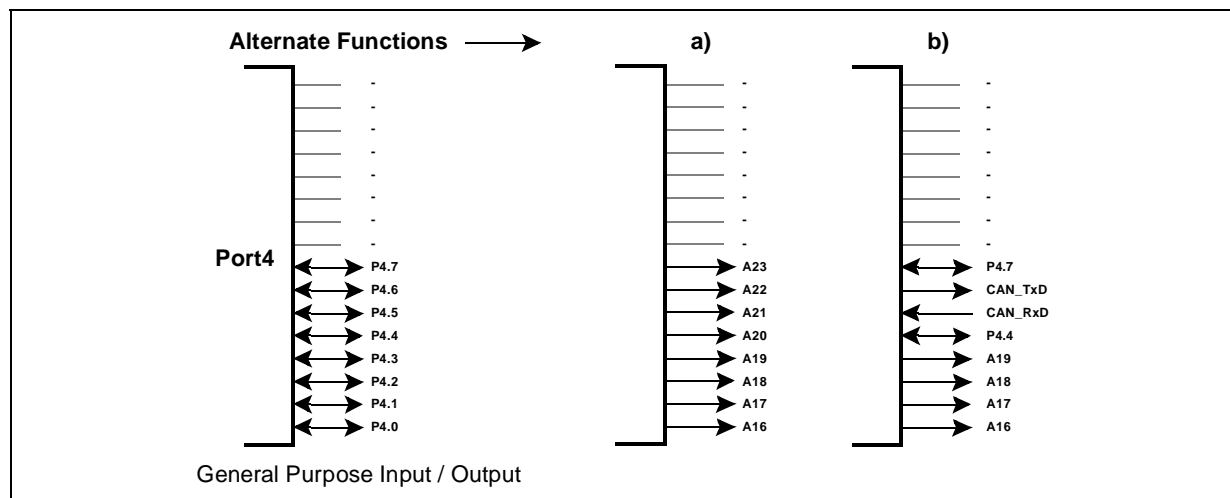
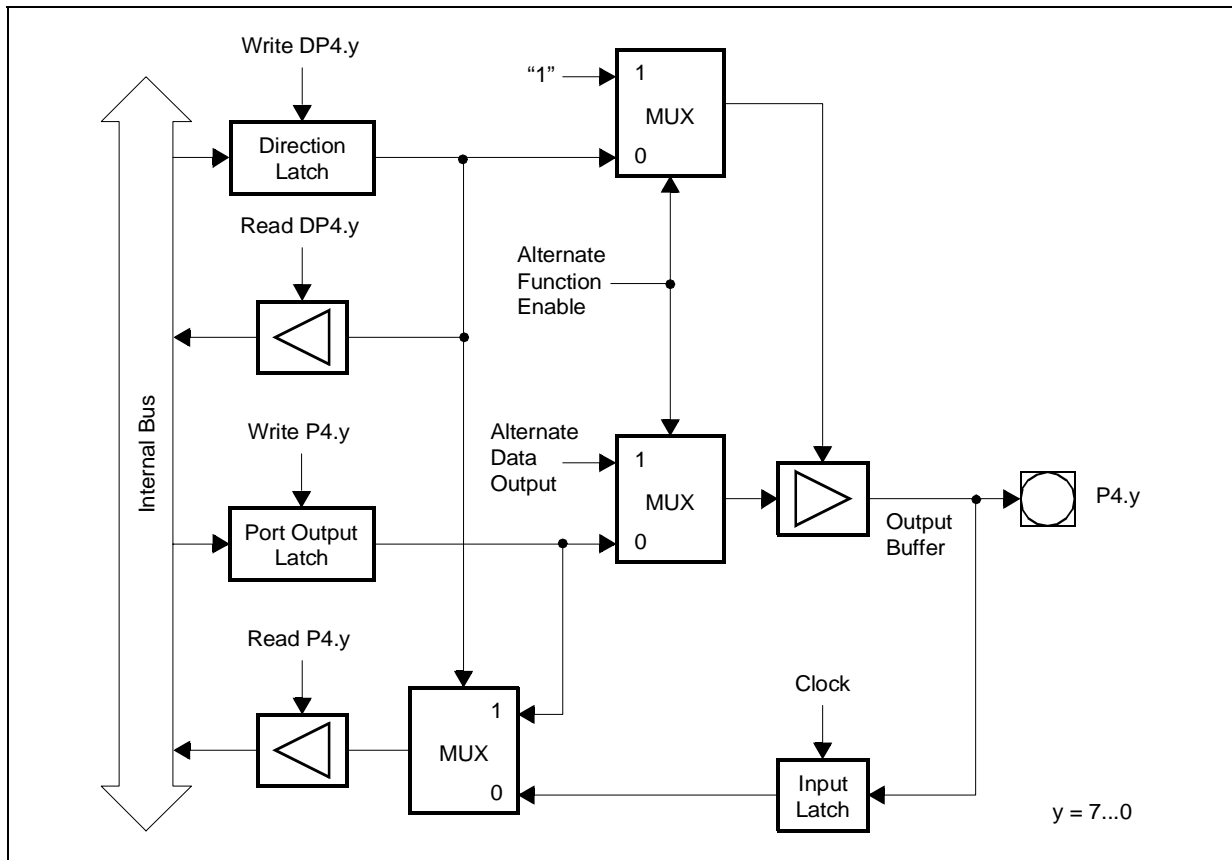


Figure 36 : Block diagram of a Port4 pin



6.7 - 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)

### 6.7.1 - Alternate Functions of Port5

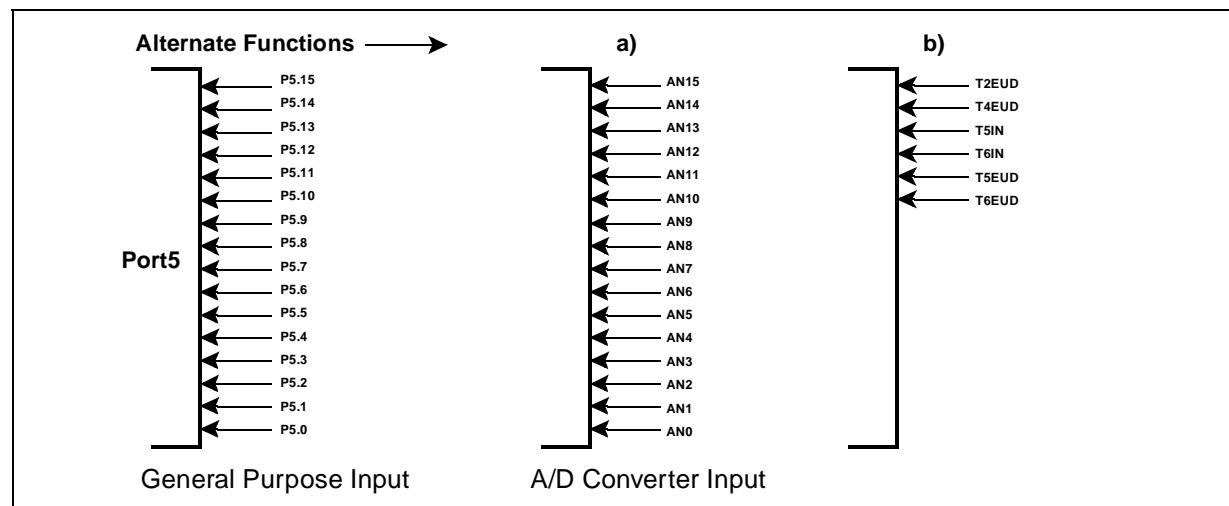
Each line of Port5 is also connected to the input multiplexer of the Analog/Digital Converter. All port lines (P5.15...P5.0) can accept analog signals (AN15...AN0) that can be converted by the ADC. No special programming is required for pins that shall be used as analog inputs. The upper 6 pins of Port5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

The Table 14 summarizes the alternate functions of Port5.

**Table 14** : Port5 alternate functions

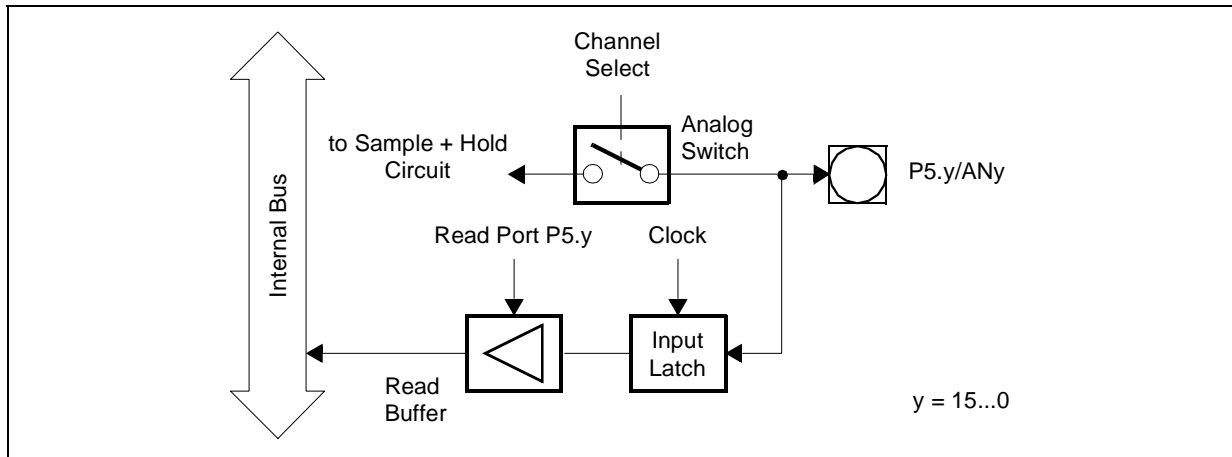
Port5 Pin	Alternate Function a)	Alternate Function b)
P5.0	Analog Input AN0	-
P5.1	Analog Input AN1	-
P5.2	Analog Input AN2	-
P5.3	Analog Input AN3	-
P5.4	Analog Input AN4	-
P5.5	Analog Input AN5	-
P5.6	Analog Input AN6	-
P5.7	Analog Input AN7	-
P5.8	Analog Input AN8	-
P5.9	Analog Input AN9	-
P5.10	Analog 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 37** : Port5 I/O and alternate functions



Port5 pins have a special port structure (see Figure 38), 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 38** : Block diagram of a Port5 pin



**6.8 - 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	<b>Port direction register DP6 Bit y</b> 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	<b>Port6 Open Drain control register Bit y</b> 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

6.8.1 - 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 ST10X167 system.

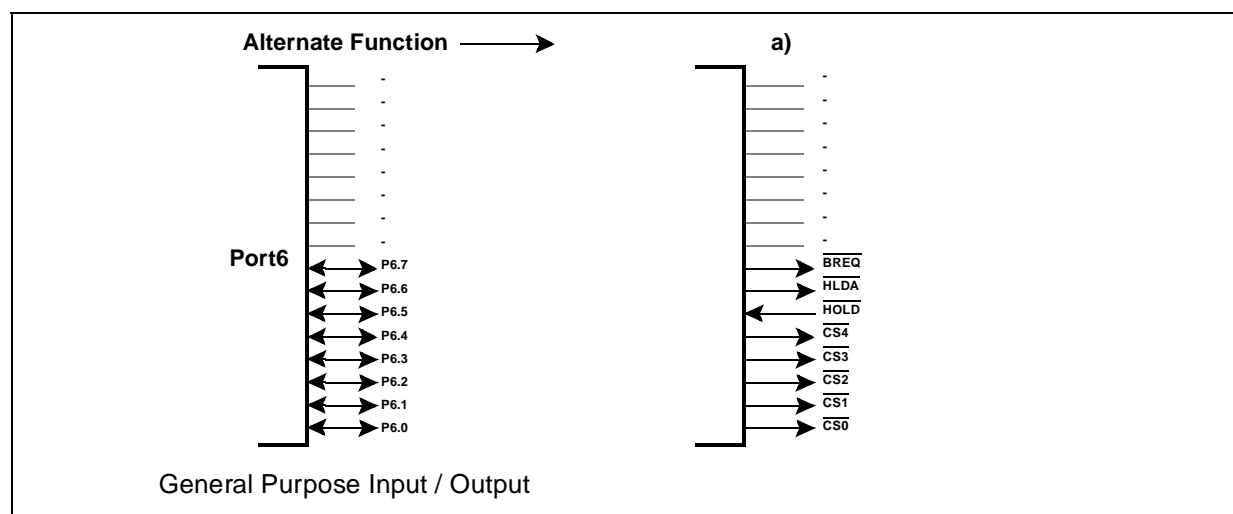
The number of chip select signals is selected via PORT0 during reset. The selected value can be read from Bitfield CSSEL in register RP0H (read only) in order to check the configuration during run time.

The Table 15 summarizes the alternate functions of Port6 depending on the number of selected chip select lines (coded via Bitfield CSSEL).

Table 15 : 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 39 : 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 ST10X167 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 ST10X167 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 40).

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 41).

**Figure 40** : Block diagram of Port6 Pins with an alternate output function

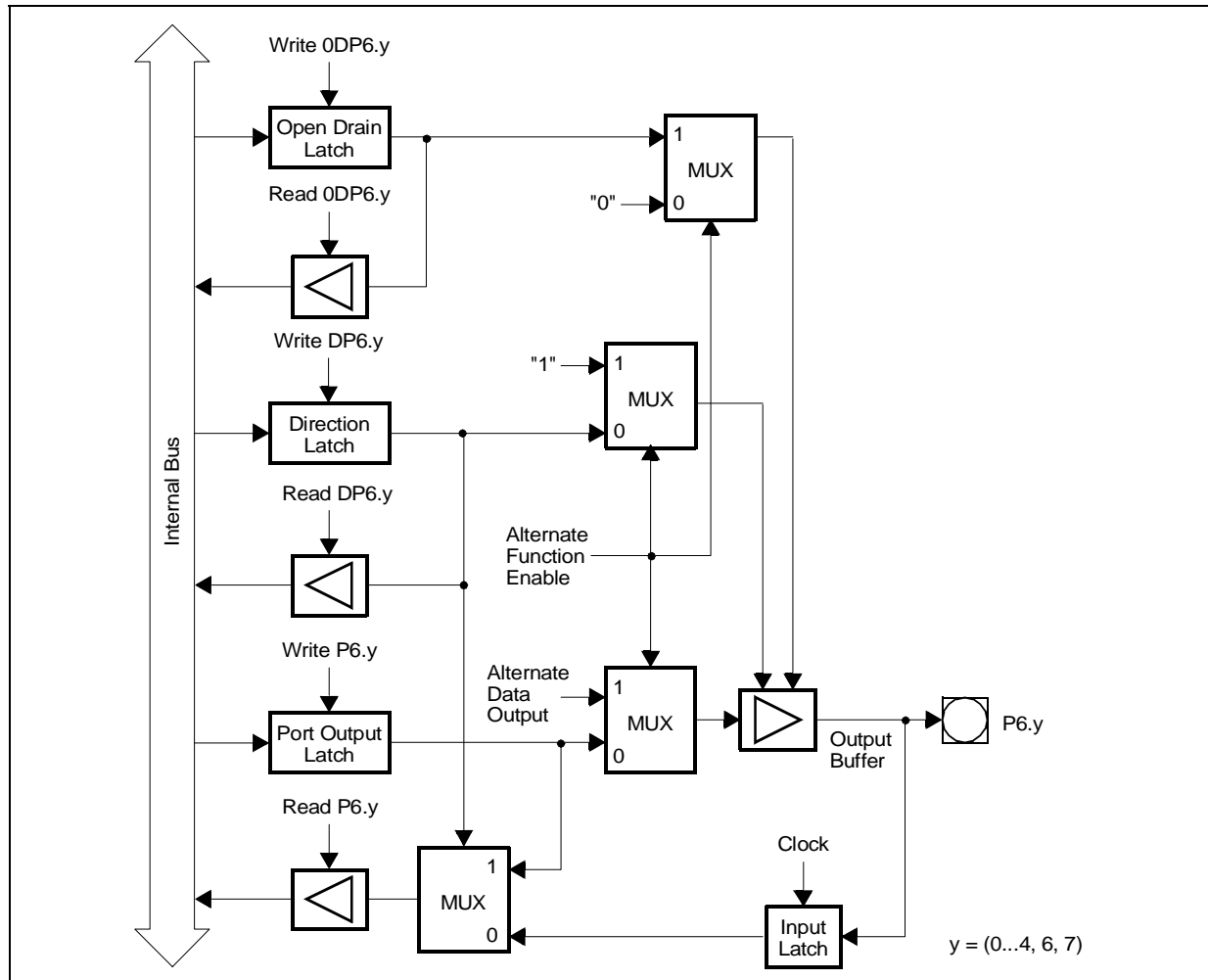
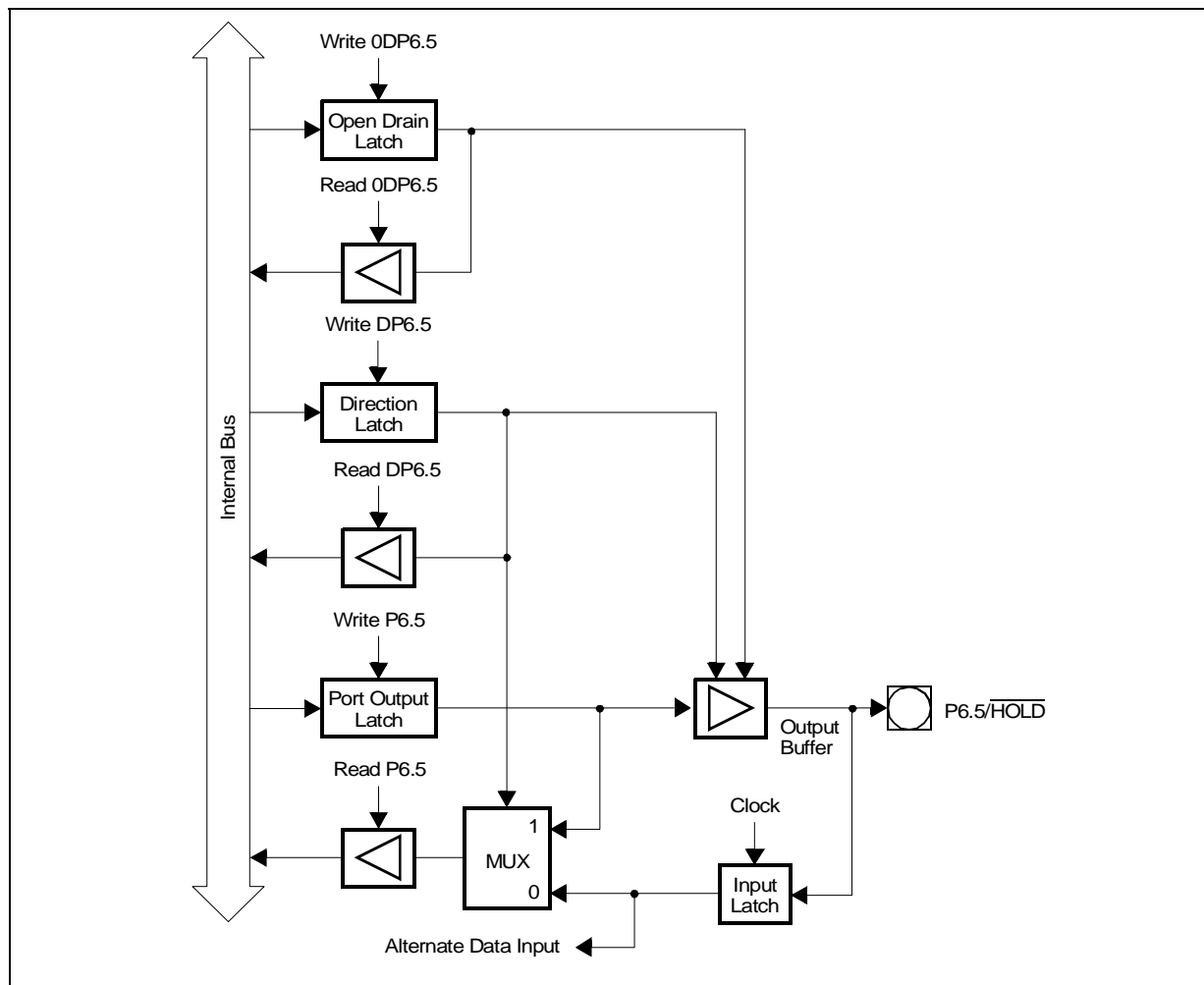


Figure 41 : Block diagram of Pin P6.5 ( $\overline{\text{HOLD}}$ )



### 6.9 - 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
-	-	-	-	-	-	-	-	<b>P7.7</b>	<b>P7.6</b>	<b>P7.5</b>	<b>P7.4</b>	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

<b>DP7 (FFD2h / E9h)</b>								<b>SFR</b>								Reset Value: --00h															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	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	-	-	-	-	-	-	-	-	ODP7.7	ODP7.6	ODP7.5	ODP7.4	ODP7.3	ODP7.2	ODP7.1	ODP7.0
								RW																							

Bit	Function
DP7.y	<b>Port direction register DP7 Bit y</b> DP7.y = 0: Port line P7.y is an input (high-impedance) DP7.y = 1: Port line P7.y is an output

<b>ODP7 (F1D2h / E9h)</b>								<b>ESFR</b>								Reset Value: --00h															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	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	-	-	-	-	-	-	-	-	ODP7.7	ODP7.6	ODP7.5	ODP7.4	ODP7.3	ODP7.2	ODP7.1	ODP7.0
								RW																							

Bit	Function
ODP7.y	<b>Port7 Open Drain control register Bit y</b> 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

**6.9.1 - 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.

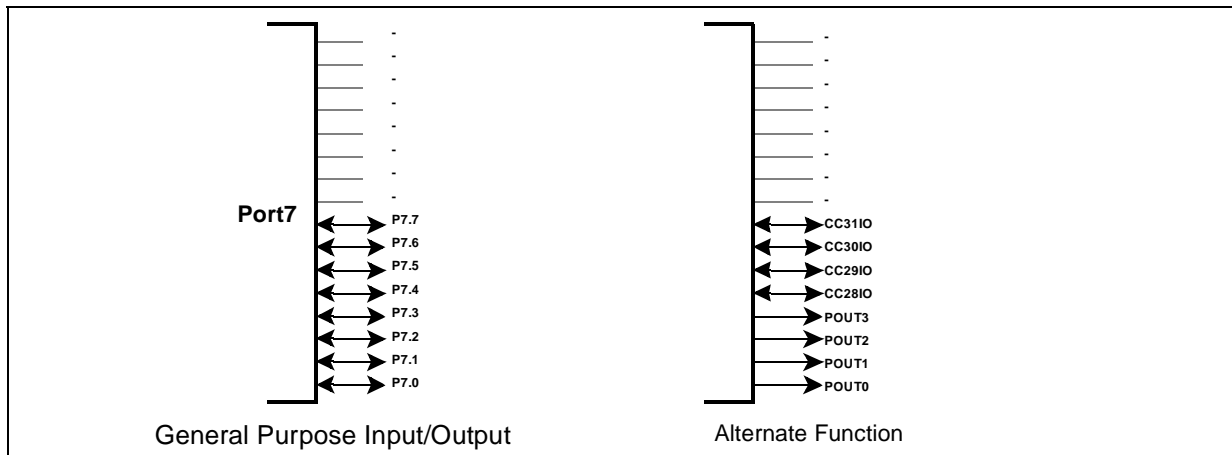
The Table 16 summarizes the alternate functions of Port7.

**Table 16 :** 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



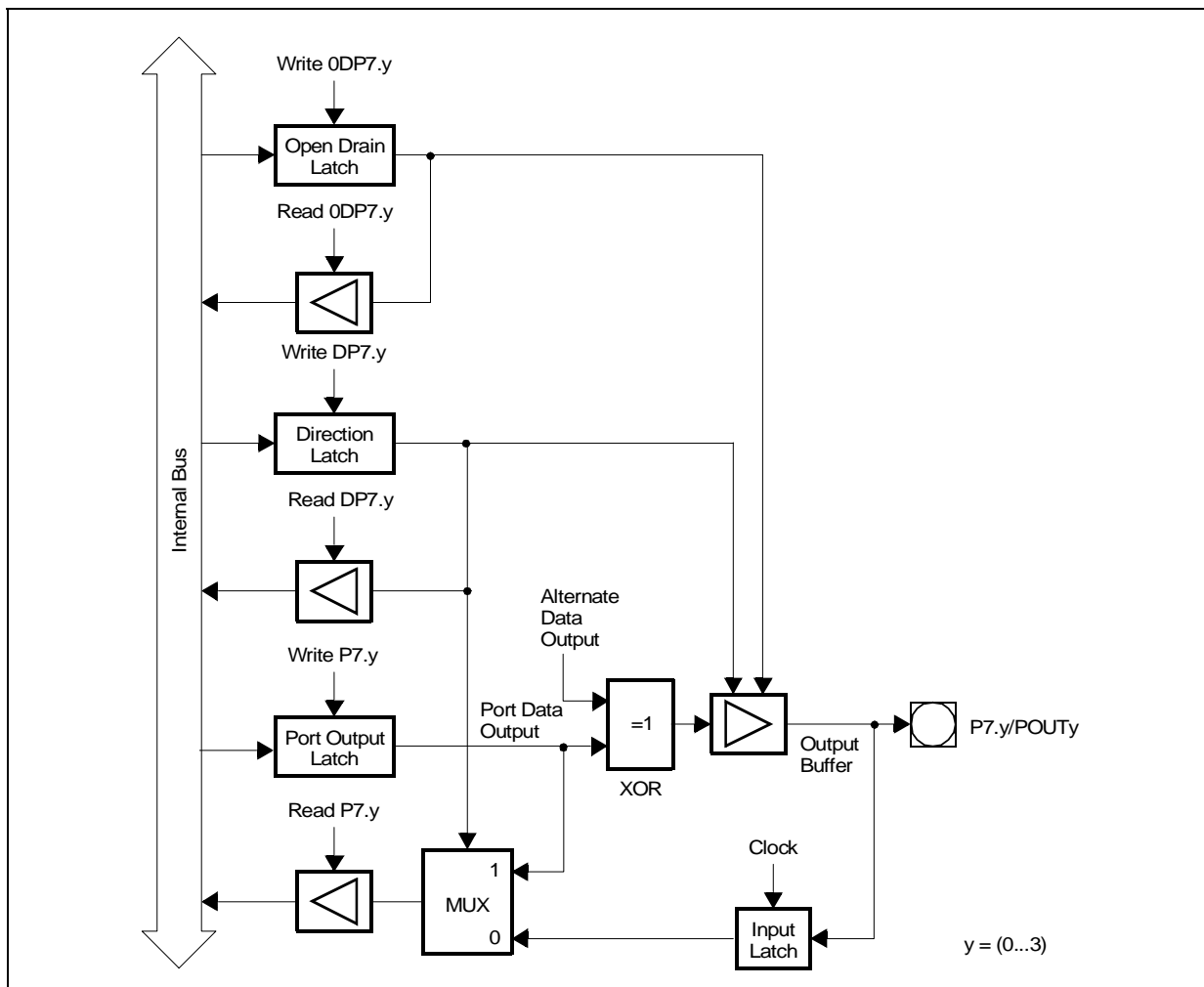
Figure 42 : Port7 I/O and alternate functions



The structure of Port7 differ in the way the output latches are connected to the internal bus and to the pin driver (see Figure 43 and Figure 44).

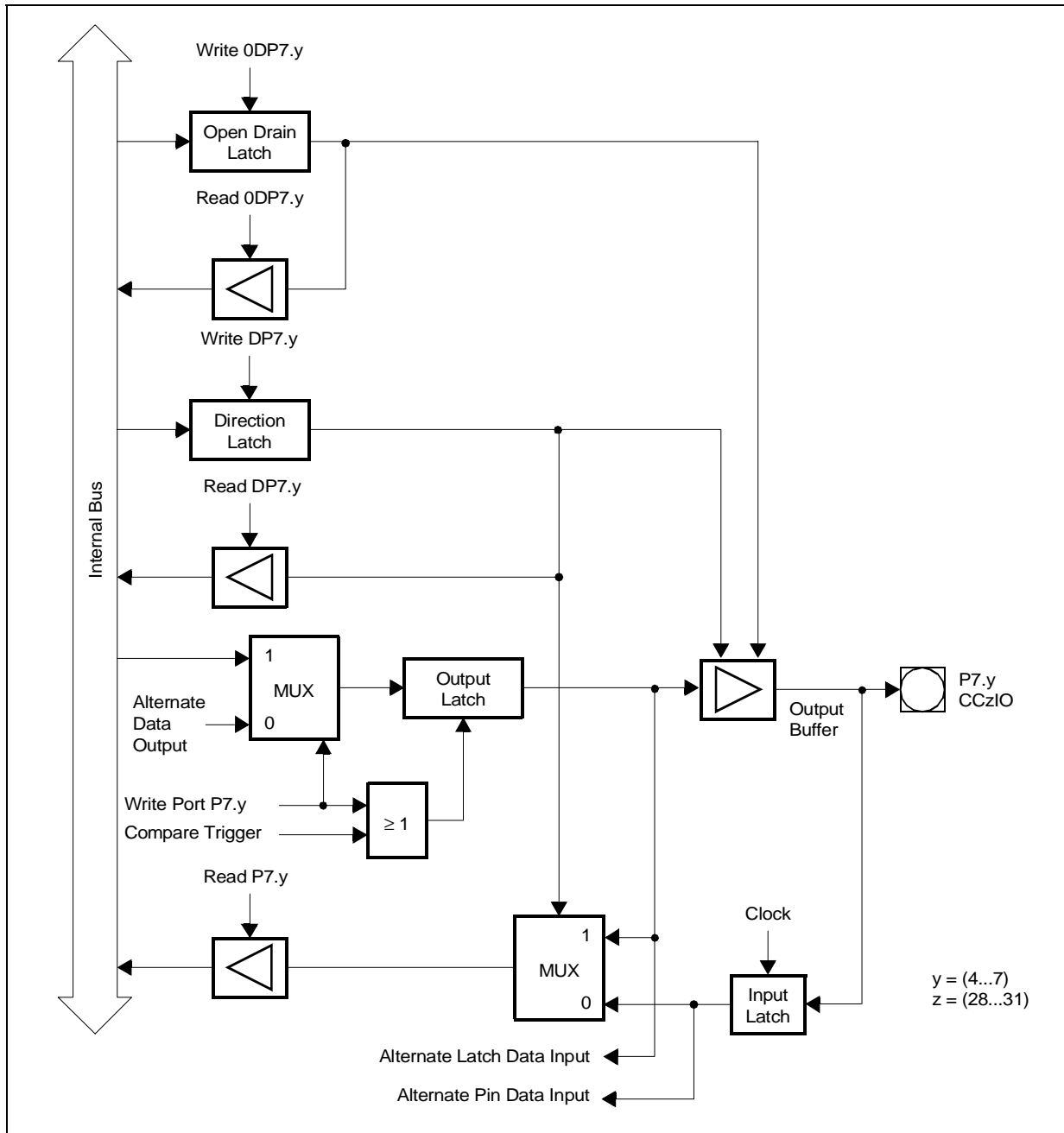
Pins P7.3...P7.0 (POUT3...POUT0) XOR the alternate data output with the port latch output, which allows to use the alternate data directly or inverted at the pin driver.

Figure 43 : 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 44 : Block diagram of Port7 pins P7.7...P7.4



### 6.10 - 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	<b>Port direction register DP8 Bit y</b> 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	<b>Port8 Open Drain control register Bit y</b> 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

**6.10.1 - 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 17). The use of the port lines by the CAPCOM unit, its accessibility

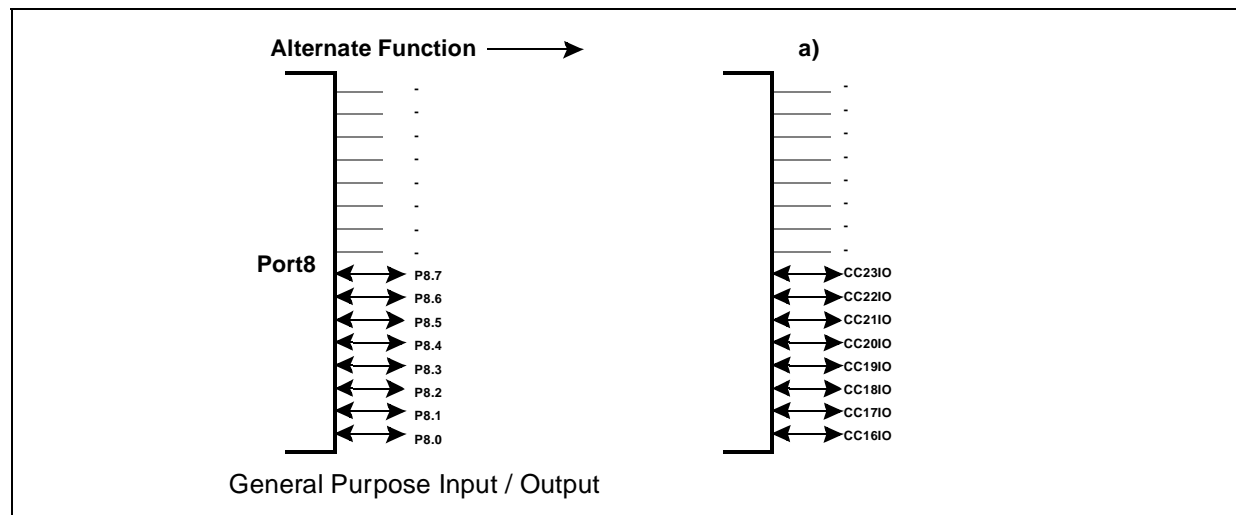
via software and the precautions are the same as described for the Port2 lines.

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 17** : Port8 alternate functions

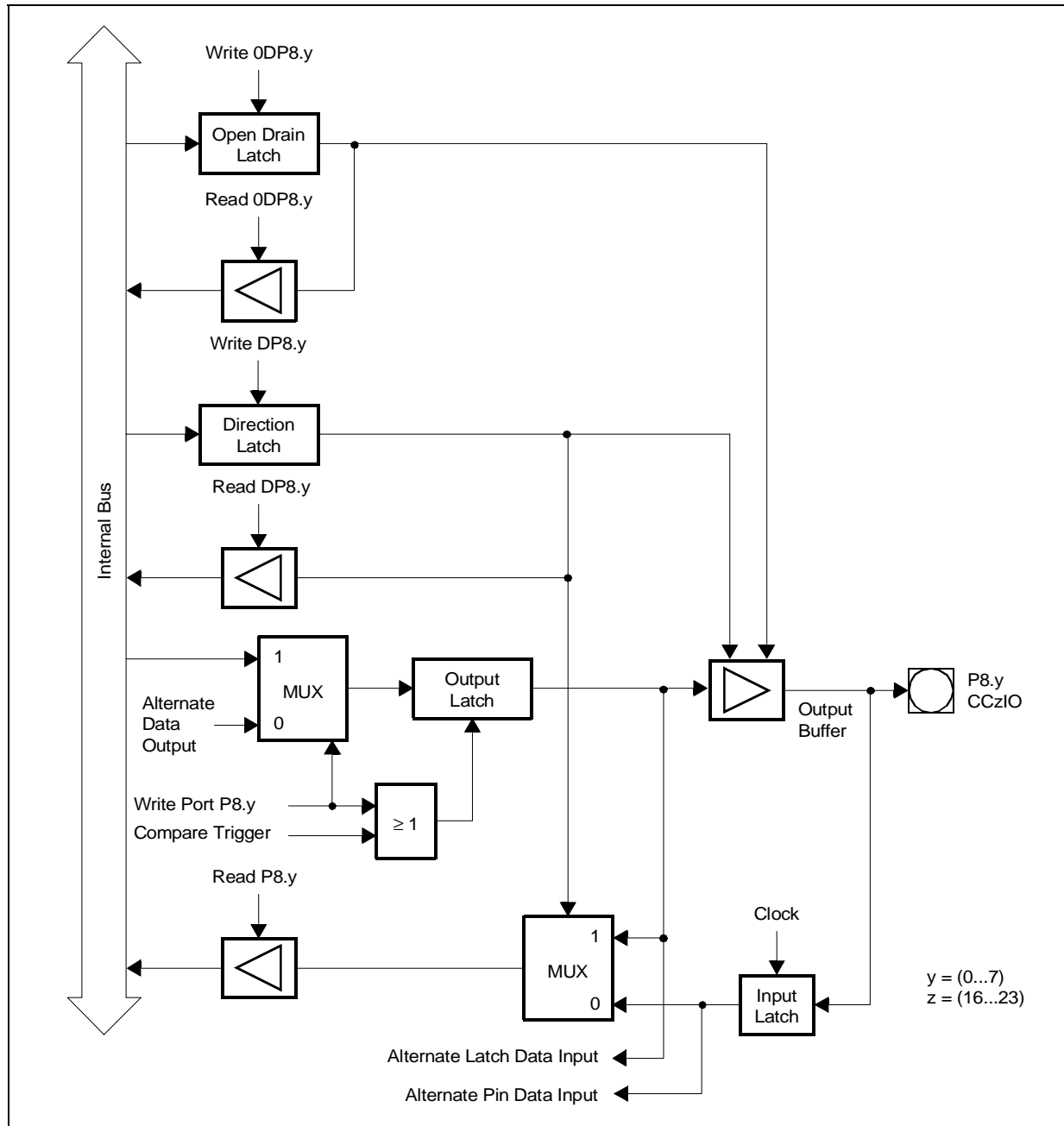
Port8 Pin	Alternate Function
P8.0	CC16IO Capture input / compare output channel 16
P8.1	CC17IO Capture input / compare output channel 17
P8.2	CC18IO Capture input / compare output channel 18
P8.3	CC19IO Capture input / compare output channel 19
P8.4	CC20IO Capture input / compare output channel 20
P8.5	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 45** : 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 46** : Block diagram of Port8 pins



7 - DEDICATED PINS

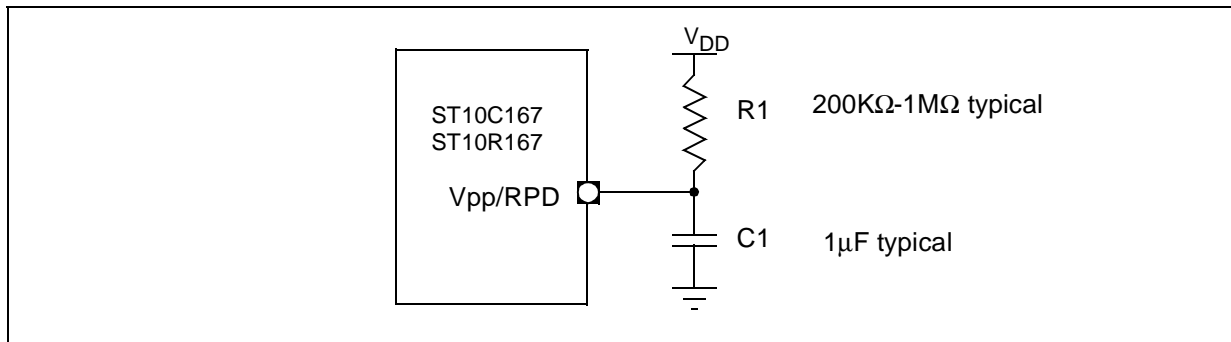
Most of the input/output or control signals of the ST10X167 are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins,

including the oscillator, special control signals and the power supply.

The Table 18 summarizes the dedicated pins of the ST10X167.

Table 18 : 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 ROM, 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 ST10X167 reads data from these external devices. During reset and during Hold mode an internal pull-up ensures an inactive high level on the RD output.
$\overline{WR/WRL}$	External Write/Write Low Strobe: controls the data transfer from the ST10X167 to an external memory or peripheral device. This pin may either provide a general WR signal activated for both Byte and Word write accesses, or specifically control the low Byte of an external 16 Bit device (WRL) together with the signal WRH (alternate function of P3.12/BHE). During reset and during Hold mode an internal pull-up ensures an inactive (high) level on the WR/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 ST10F167 the polarity is always $\overline{READY}$ . For the ST10C167 and ST10R167 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 ST10X167 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 ST10X167 into power-down mode.
$\overline{RSTIN}$	Reset Input: puts the ST10X167 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 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.
$V_{DD}$ , $V_{SS}$	Digital Power Supply and Ground (6 pins each): provides the power supply for the digital logic of the ST10X167. All $V_{DD}$ pins and all $V_{SS}$ pins must be connected to the power supply and ground, respectively.
$V_{PP}$	Flash Programming Voltage for ST10F167 or Exit from powerdown for ST10C167 and ST10R167 devices: 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 Vpp 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 47. For more information on exiting power down mode refer to Chapter 19 - Power Reduction Modes.

**Figure 47** : Vpp external RC circuit for ST10C167 and ST10R167

Vpp external RC circuit is used for exiting power-down mode with external interrupt and for power-up asynchronous reset.

## 8 - THE EXTERNAL BUS INTERFACE

The on-chip peripherals and on-chip RAM and ROM / Flash Memory only cover a small fraction of the ST10X167 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 48).

Accesses to external memory or peripherals are executed by the integrated External Bus Controller (EBC). The function of the EBC is controlled via the SYSCON register and the BUSCONx and ADDRSELx registers. The BUSCONx registers specify the external bus cycles in terms of address (mux/demux), data (16 Bit/8 Bit), chip selects and length (waitstates / 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.

### 8.1 - 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.

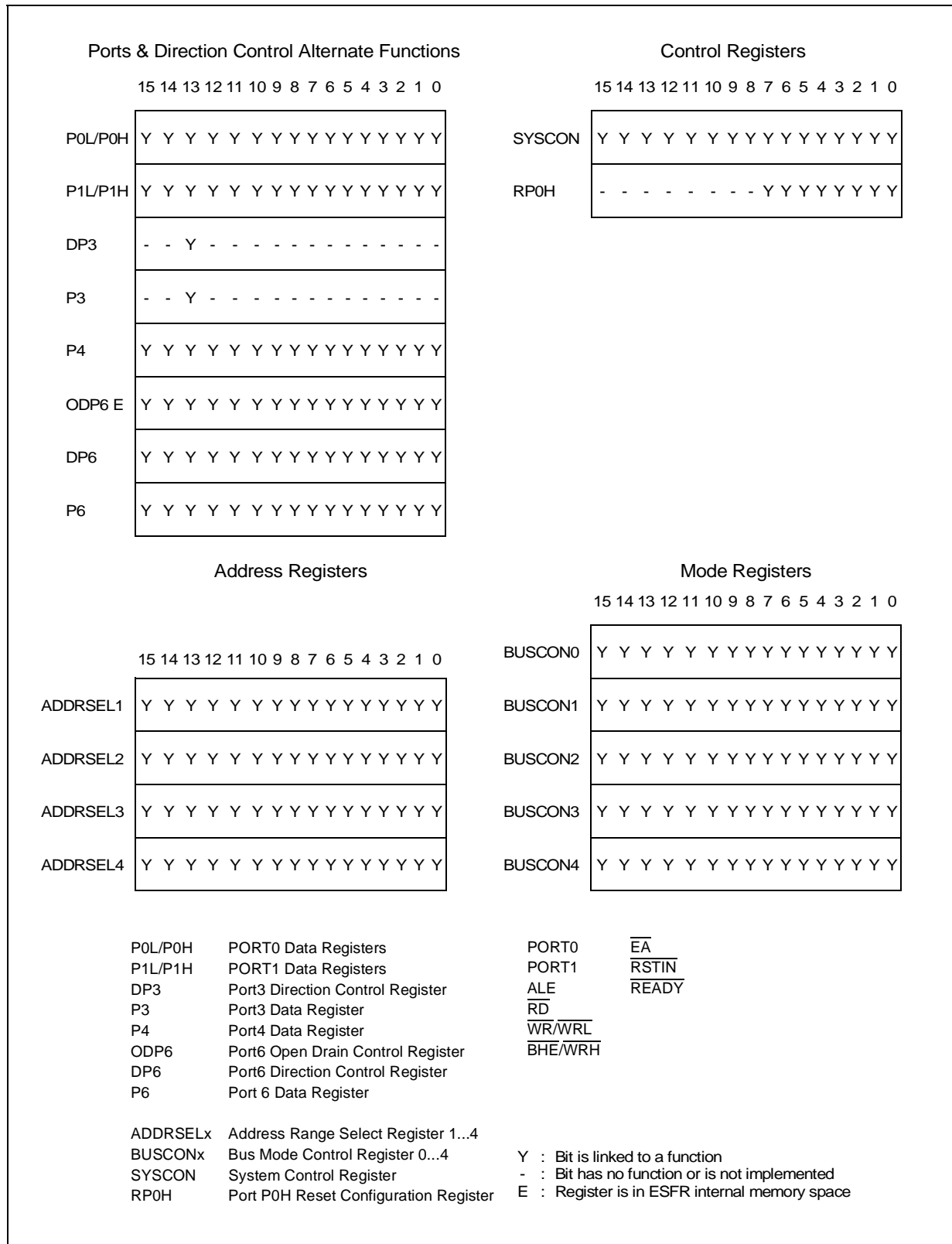
In single chip mode the ST10X167 operates only with and out of internal resources. No external bus is configured and no external peripherals and/or memory can be accessed. Also no port lines are occupied for the bus interface.

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 ST10X167 to start execution out of the internal program memory (Masked ROM / Flash Memory).

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



Figure 48 : SFRs and port pins associated with the external bus interface



## 8.2 - External Bus Modes

When the external bus interface is enabled (Bit BUSACTx='1') and configured (Bitfield BTYP), the ST10X167 uses a subset of its port lines together with some control lines to build the external bus.

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

The bus configuration (BTYP) for the address windows (BUSCON4...BUSCON1) is selected 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 EA is low during reset. Otherwise BUSCON0 may be programmed via software just like the other BUSCON registers.

The 16M Byte address space of the ST10X167 is divided into 256 segments of 64K Byte each. The 16 Bit intra-segment address is output on PORT0 for multiplexed bus modes or on PORT1 for demultiplexed bus modes.

When segmentation is disabled, only one 64K Byte 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 Bitfields SALSEL and CSSEL of register RP0H, respectively.

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

### 8.2.1 - 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.

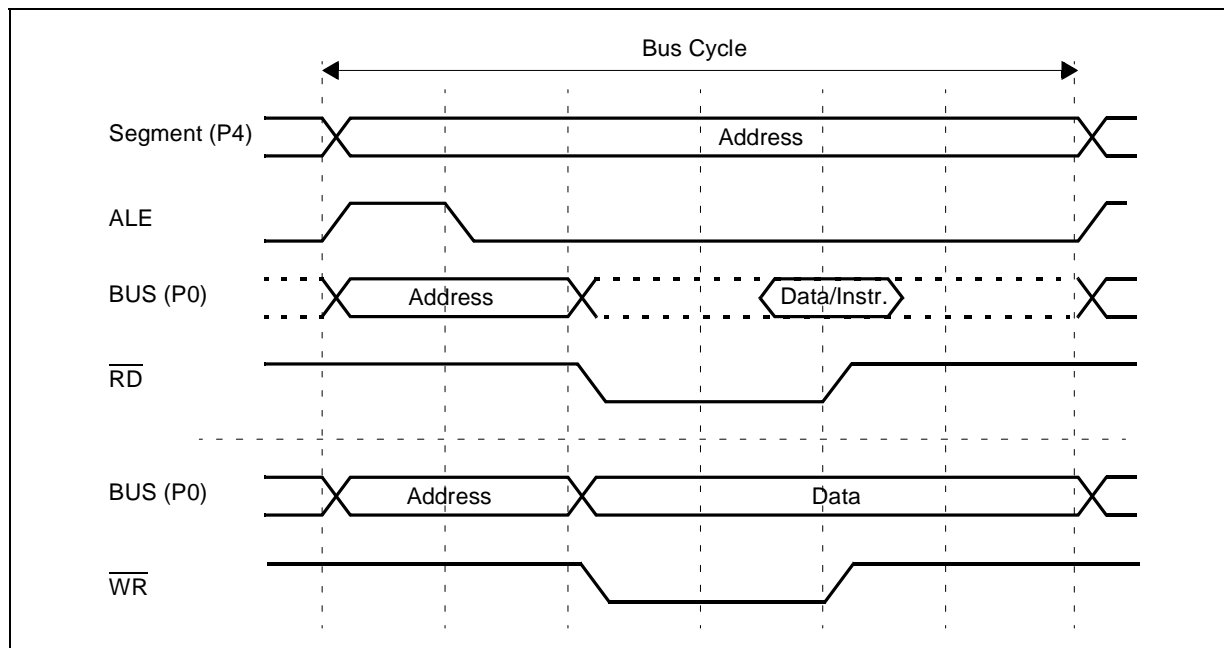
The EBC initiates an external access by generating the Address Latch Enable signal (ALE) and then placing an address on the bus. The falling edge of ALE triggers an external latch to capture the address.

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 49 : Multiplexed bus cycle



### 8.2.2 - Demultiplexed Bus Modes

In the demultiplexed bus modes the 16 Bit intra-segment address is permanently output on PORT1, while the data uses PORT0 (16 Bit data) or P0L (8 Bit data).

The upper address lines are permanently output on Port4 (if selected via SALSEL during reset). No address latches are required.

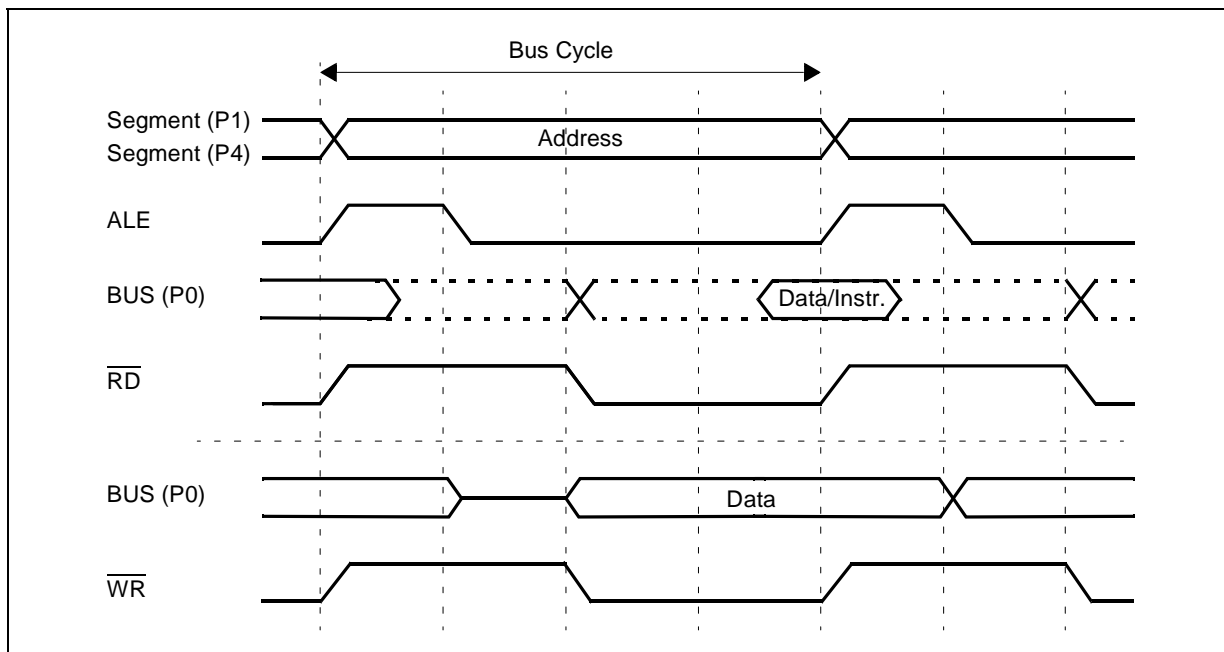
The EBC initiates an external access by placing an address on the address bus. After a programmable period of time the EBC activates the respective command signal (RD, WR, WRL, WRH).

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 50 : Demultiplexed bus cycle



### 8.2.3 - 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 demultiplexed buses or use READY control or predefined waitstates.

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

- **Reprogramming the BUSCON and/or ADDRSEL registers** allows to either change the bus mode for a given address window, or change the size of an address window that uses a certain bus mode. Reprogramming allows to use a great number of different address windows (more than BUSCONs are available) on the expense of the overhead for changing the registers and keeping appropriate tables.
- **Switching between predefined address windows** automatically selects the bus mode that is associated with the respective window. Predefined address windows allow to use different bus modes without any overhead, but restrict their number to the number of BUSCONs. However, as BUSCON0 controls all address areas, which are not covered by the other BUSCONs, this allows to have gaps between these windows, which use the bus mode of BUSCON0.

PORT1 will output the intra-segment address when any of the BUSCON registers selects a demultiplexed bus mode, even if the current bus cycle uses a multiplexed bus mode. This 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 demultiplexed 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 demultiplexed bus. However, in the multiplexed bus modes the address is also required on PORT0. In this special case the address on PORT0 is delayed by one CPU clock cycle, which delays the complete (multiplexed) bus cycle and extends the corresponding ALE signal (see Figure 51).

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

#### 8.2.4 - 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

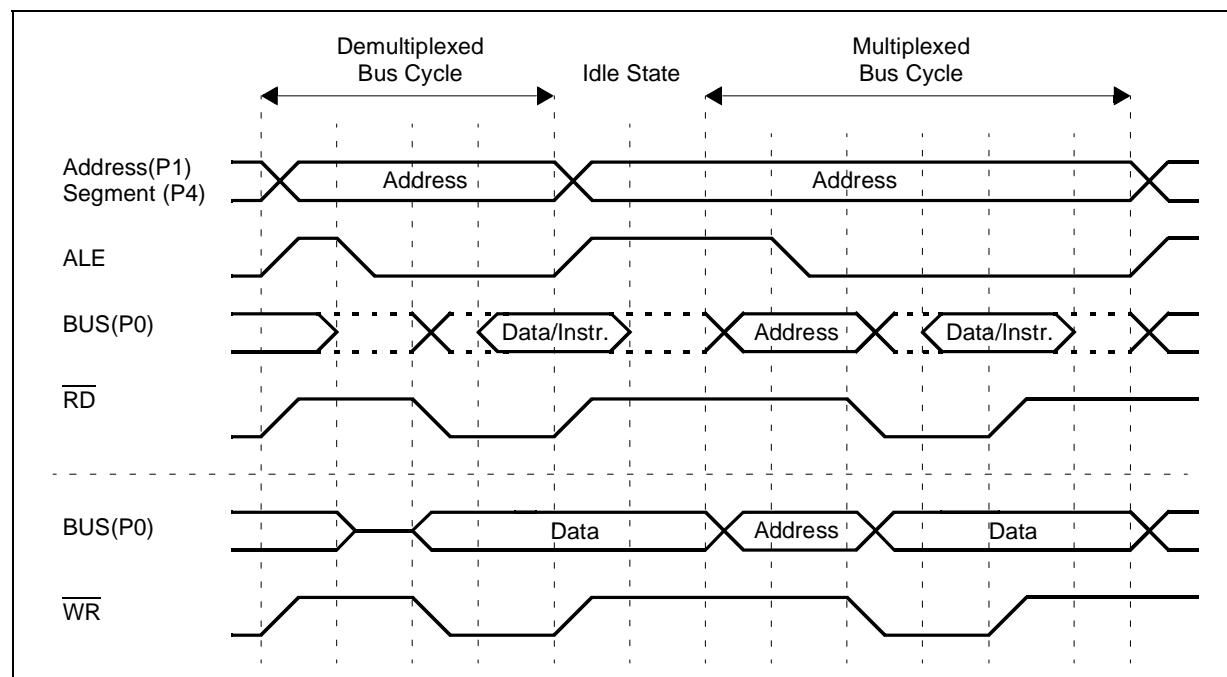
handled by the EBC and is transparent to the CPU and the programmer.

**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 BHE signal, while the lower Byte is selected with the A0 signal. So the two Byte 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 CS input, but two WR enable inputs (for the two Byte), the EBC can directly generate these two write control signals. This saves the external combination of the WR signal with A0 or BHE. In this case pin WR serves as WRL (write low Byte) and pin BHE serves as WRH (write high Byte). Bit WRCFG in register SYSCON selects the operating mode for pins WR and 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 ST10X167 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 BHE and A0.

**Figure 51** : Switching from demultiplexed 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 Demultipl.	Low (1 / 2 / 4)	Very low (no latch, Byte bus)	P0H
16 Bit Multiplexed	High (1.5 / 1.5 / 3)	High (16 Bit latch, Word bus)	P1H, P1L
16 Bit Demultipl.	Very high (1 / 1 / 2)	Low (no latch, Word bus)	---

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

### 8.2.5 - Disable / Enable Control for Pin BHE (BYTDIS)

Bit BYTDIS is provided for controlling the active low Byte High Enable (BHE) pin. The function of the 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 BHE pin is implicitly used by the External Bus Controller to select one of two Byte-organized memory chips, which are connected to the ST10X167 via a Word-wide external data bus. After reset the 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 BHE signal is not used.

### 8.2.6 - 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 RPOH (see table below).

SALSEL	Segment address lines	Directly accessible address space
1 1	Two: A17...A16	256K Byte (Default without pull-downs)
1 0	Eight: A23...A16	16M Byte (Maximum)
0 1	None	64K Byte (Minimum)
0 0	Four: A19...A16	1M Byte

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

### 8.2.7 - CS Signal Generation

During external accesses the EBC can generate a (programmable) number of CS lines on Port6, which allows to directly select external peripherals

or memory banks without requiring an external decoder. The number of CS lines is selected during reset and coded in Bit field CSSEL in register RPOH (see table below).

CSSEL	Chip Select Lines	Note
1 1	Five: CS4...CS0	Default without pull-downs
1 0	None	Port6 pins free for I/O
0 1	Two: CS1...CS0	
0 0	Three: CS2...CS0	

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

Note The 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 CS signals. Upon accesses to address windows without a selected CS line all selected 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 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.

### 8.2.8 - Segment Address Versus Chip Select

Note : This feature is not available for the ST10F167.

The external bus interface supports many configurations for the external memory. By increasing the number of segment address lines, a linear address space of 256K Byte, 1M Byte or 16M Byte 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 1M Byte each, so the available address space is 5M Byte (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).

### 8.3 - Programmable Bus Characteristics

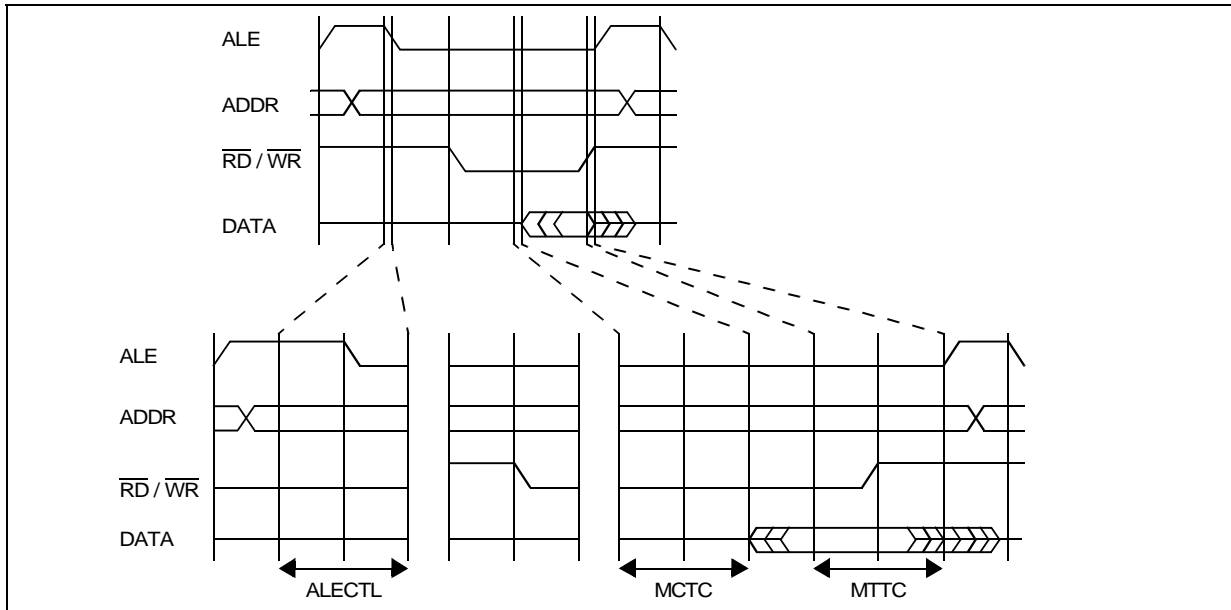
Important timing characteristics of the external bus interface have been made user programmable to allow to adapt it to a wide range of different external bus and memory configurations with different types of memories and/or peripherals.

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 waitstates) defines the allowable access time
- **Memory tri-state time** (extendable with 1 waitstate) defines the time for a data driver to float
- **Read/write delay time** defines when a command is activated after the falling edge of ALE
- **READY polarity** is programmable for ST10C167 and ST10R167 only
- **READY control** defines, if a bus cycle is terminated internally or externally
- **Programmable chip select timing control** for ST10C167 and ST10R167 only

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 52 : Programmable external bus cycle



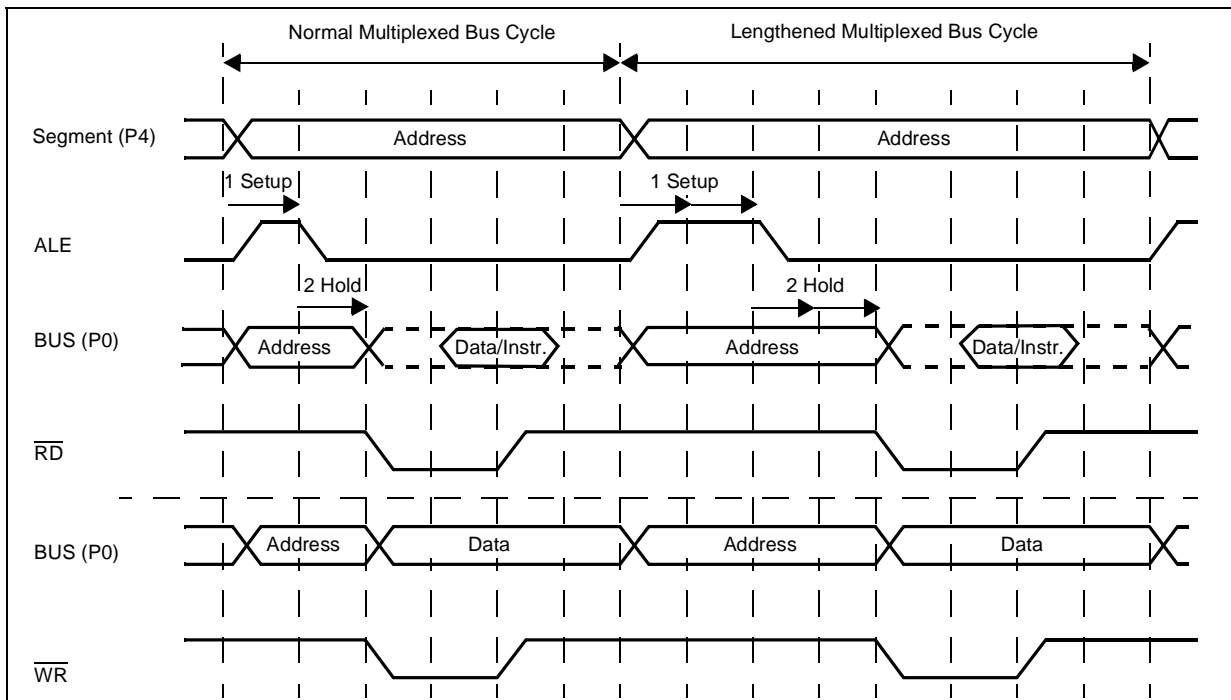
8.3.1 - ALE Length Control

The length of the ALE signal and the address hold time after its falling edge are controlled by the ALECTLx 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 53 : ALE length control



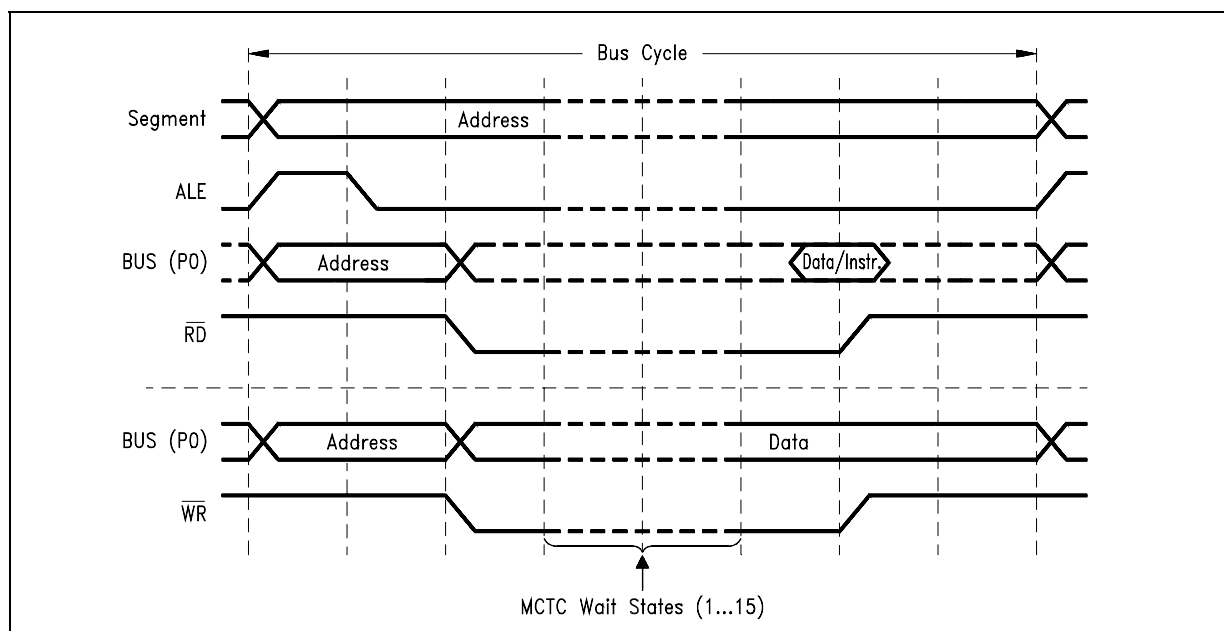


### 8.3.2 - Programmable Memory Cycle Time

The ST10X167 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 ST10X167 can be extended for a memory or a peripheral, which cannot keep pace with the controller's maximum speed some waitstates are introduced during the access (see Figure ). During these memory cycle time waitstates, the CPU is idle, if this access is required for the execution of the current instruction. The memory cycle time waitstates can be programmed in increments of one CPU clock within a range from 0 to 15 (default after reset) via the MCTC fields of the BUSCON registers. 15-(MCTC) waitstates will be inserted.

**Figure 54** : Memory cycle time



### 8.3.3 - Programmable Memory Tri-state Time

The ST10X167 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 waitstate after the previous bus cycle (see Figure 55).

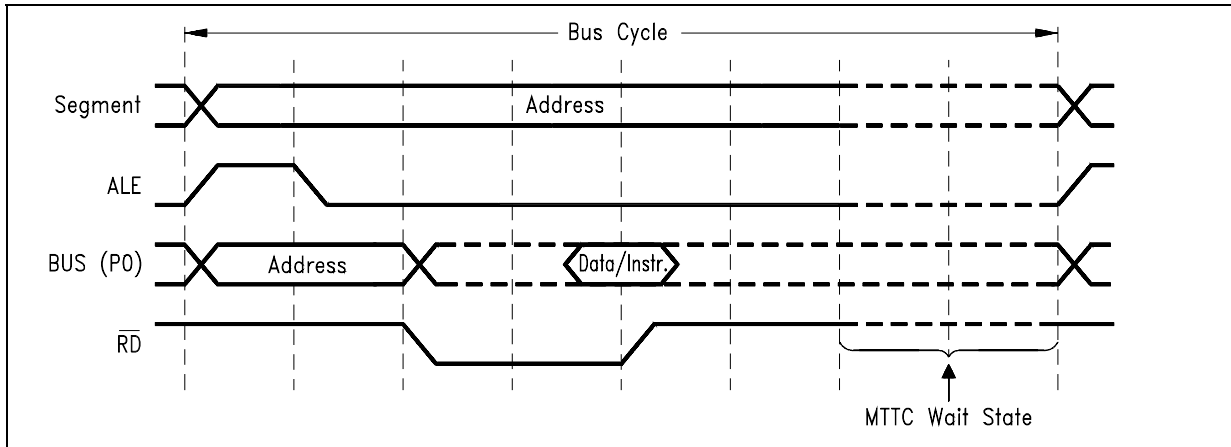
During this memory tri-state time waitstate, the CPU is not idle, so CPU operations will only be slowed down if a subsequent external instruction or data fetch operation is required during the next instruction cycle.

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

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

Any MTTC waitstates are applicable to both read and write cycles.

Figure 55 : Memory tri-state time



8.3.4 - Read / Write Signal Delay

The ST10X167 allows the user to adjust the timing of the read and write commands to account for timing requirements of external peripherals.

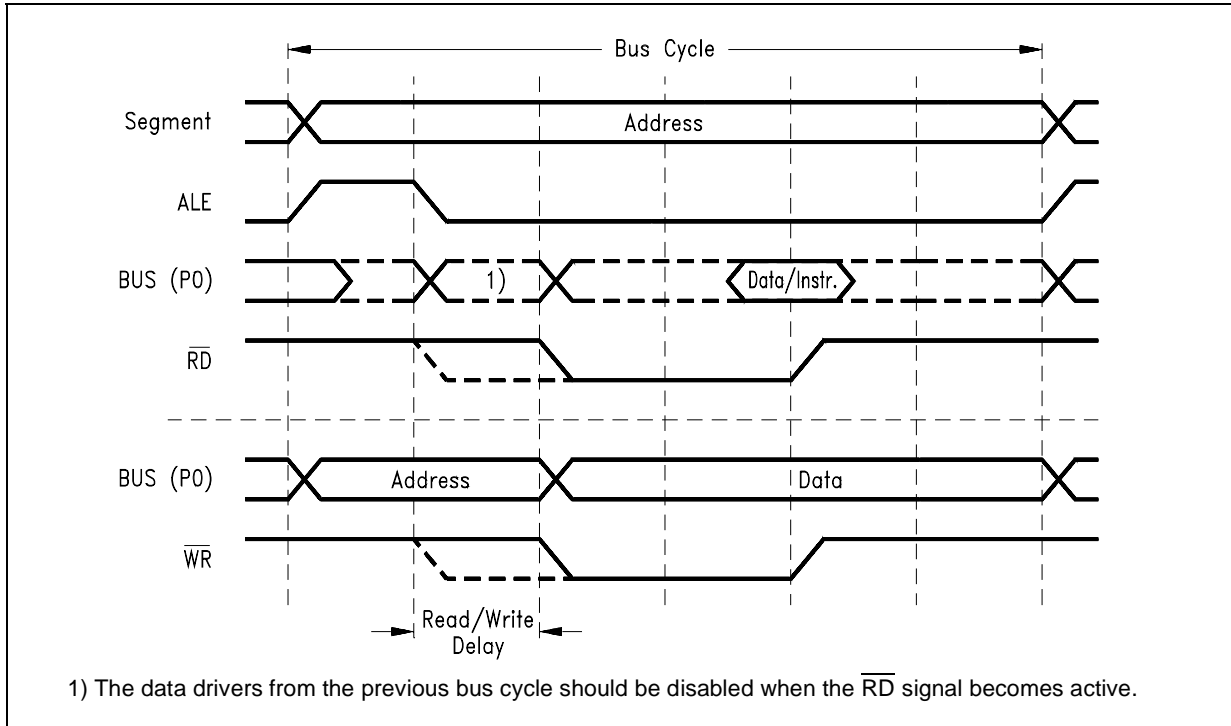
The read/write delay controls the time between the falling edge of ALE and the falling edge of the command. Without read/write delay the falling edges of ALE and command(s) are coincident (except for propagation delays). With the delay enabled, the command(s) become active half a CPU clock cycle after 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 ST10X167's address, when the early 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 56 : Read/write delay



1) The data drivers from the previous bus cycle should be disabled when the RD signal becomes active.

### 8.3.5 - READY Polarity

For the ST10C167 and ST10R167, the active level of the ready pin can be set to READY or  $\overline{\text{READY}}$  by the RDYPOL Bit 13 in the BUSCON register. For the ST10F167 the active level is fixed on  $\overline{\text{READY}}$ .

### 8.3.6 - READY / $\overline{\text{READY}}$ Controlled Bus Cycles

For ST10C167 and ST10R167 the active level of the ready pin can be set to READY or  $\overline{\text{READY}}$  by the RDYPOL Bit in the BUSCON register.

For situations where the programmable waitstates are not enough, or where the response (access) time of a peripheral is not constant, the ST10X167 provides external bus cycles that are terminated by a READY or  $\overline{\text{READY}}$  input signal (synchronous or asynchronous). In this case the ST10X167 first inserts a programmable number of waitstates (0...7) and then monitors the READY or  $\overline{\text{READY}}$  line to determine the actual end of the current bus cycle. The external device drives READY or  $\overline{\text{READY}}$  low in order to indicate that data has been latched (write cycle) or are available (read cycle).

For the ST10C167 and ST10R167, 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 57).

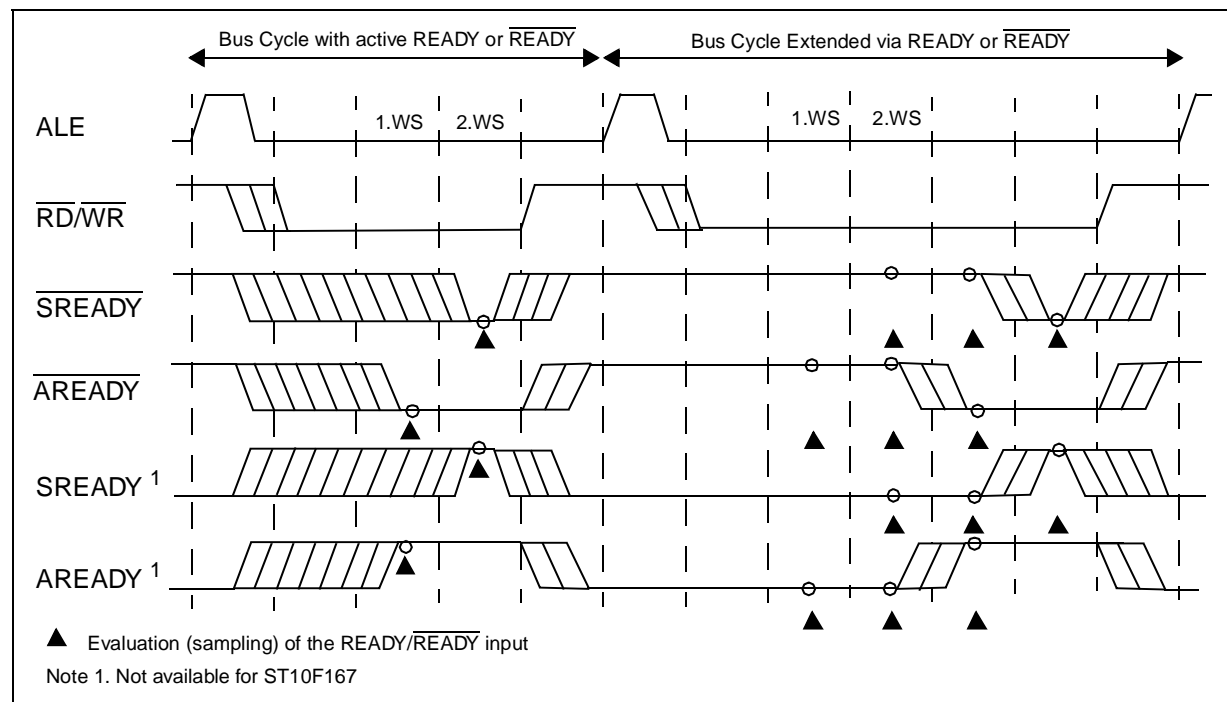
The  $\overline{\text{READY}}/\overline{\text{READY}}$  function is enabled by the RDYENx Bit in the BUSCON registers. When this function is selected (RDYENx = '1'), only the lower 3 Bit of the respective MCTC Bit field define the number of inserted waitstates (0...7), while the MSB of Bit field MCTC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

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

**The synchronous  $\overline{\text{READY}}/\overline{\text{READY}}$  (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  $\overline{\text{READY}}/\overline{\text{READY}}$  timing in this case.

**The asynchronous  $\overline{\text{READY}}/\overline{\text{READY}}$  (AREADY /  $\overline{\text{AREADY}}$ )** is less restrictive, but requires additional waitstates caused by the internal synchronization. As the asynchronous  $\overline{\text{READY}}/\overline{\text{READY}}$  is sampled earlier (see Figure 57) programmed waitstates may be necessary to provide proper bus cycles (see also notes on "normally-ready" peripherals below).

Figure 57 :  $\overline{\text{READY}}/\overline{\text{READY}}$  controlled bus cycles



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

**Note** When the  $\overline{\text{READY/READY}}$  function is enabled for a specific address window, each bus cycle within this window must be terminated with an active  $\overline{\text{READY/READY}}$  signal. Otherwise the controller hangs until the next reset. A time-out function is only provided by the watchdog timer.

**Combining the  $\overline{\text{READY/READY}}$  function with predefined waitstates** is advantageous in two cases:

- Memory components with a fixed access time and peripherals operating with  $\overline{\text{READY/READY}}$  may be grouped into the same address window. The (external) waitstate control logic in this case would activate  $\overline{\text{READY/READY}}$  either upon the memory's chip select or with the peripheral's  $\overline{\text{READY/READY}}$  output. After the predefined number of waitstates the ST10X167 will check its  $\overline{\text{READY/READY}}$  line to determine the end of the bus cycle. For a memory access it will be below already (see Figure 57), 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  $\overline{\text{READY/READY}}$  function with so-called "normally-ready" peripherals, it may lead to erroneous bus cycles, if the  $\overline{\text{READY/READY}}$  line is sampled too early. These

peripherals pull their  $\overline{\text{READY/READY}}$  output low, while they are idle. When they are accessed, they deactivate  $\overline{\text{READY/READY}}$  until the bus cycle is complete, then drive it low again. If, however, the peripheral deactivates  $\overline{\text{READY/READY}}$  after the first sample point of the ST10X167, the controller samples an active  $\overline{\text{READY/READY}}$  and terminates the current bus cycle, which, of course, is too early. By inserting predefined wait-states the first  $\overline{\text{READY/READY}}$  sample point can be shifted to a time, where the peripheral has safely controlled the  $\overline{\text{READY/READY}}$  line (after 2 wait-states in the Figure 57).

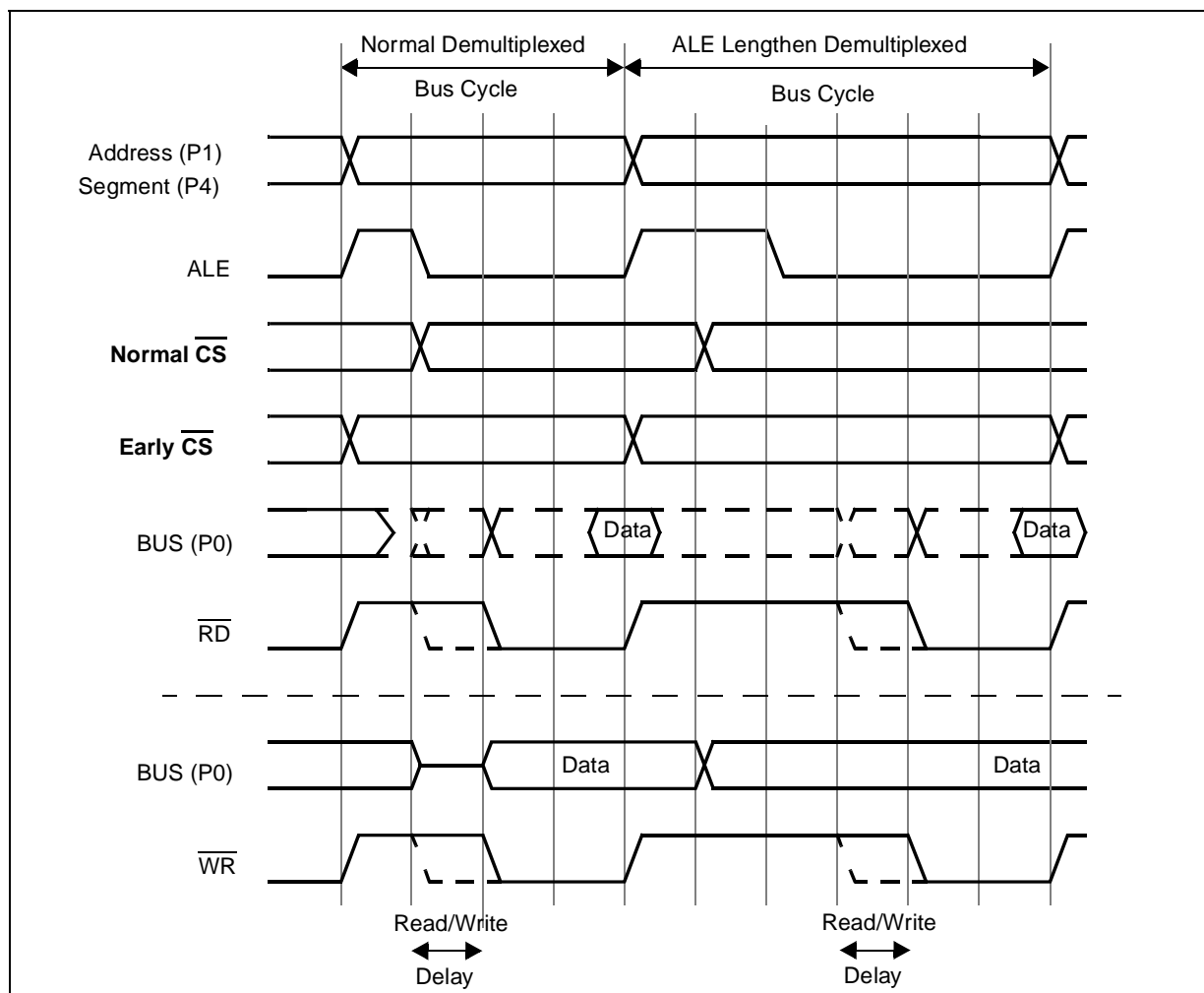
**Note** For the ST10X167 the active level of the ready pin is set to  $\overline{\text{READY}}$ . For the ST10C167 and ST10R167 the active level of the ready pin can be set to  $\overline{\text{READY}}$  or  $\text{READY}$  by the RDYPOL Bit in the BUSCON register.

### 8.3.7 - Programmable Chip Select Timing Control

The position of the  $\overline{\text{CS}}$  lines can be changed for the ST10C167 and ST10R167. 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.

This feature is not available for the ST10F167.

Figure 58 : Chip select delay



#### 8.4 - Controlling the External Bus Controller

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

The properties of a bus cycle like chip select mode, usage of  $\overline{READY}$ , length of ALE, external bus mode, read/write delay and waitstates are controlled 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.

## ST10X167

### SYSCON (FF12h / 89h)

### SFR

Reset Value: 0X00h<sup>1</sup>

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	RW	RW

Note 1. Reset value is 0XX0h for ST10F167.

Bit	Function
XPER-SHARE	<b>XBUS Peripheral Share Mode Control</b> '0': External accesses to XBUS peripherals are disabled '1': XBUS peripherals are accessible via the external bus during hold mode
VISIBLE	<b>Visible Mode Control</b> '0': Accesses to XBUS peripherals are done internally '1': XBUS peripheral accesses are made visible on the external pins
XPEN	<b>XBUS Peripheral Enable Bit</b> For ST10C167 and ST10R167 this Bit is used to enable XRAM and XCAN. For ST10F167 this Bit is used to enable XRAM only as XCAN is always enabled. '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 not allocated in ST10F167	<b>Bidirectional Reset Enable</b> '0': $\overline{\text{RSTIN}}$ pin is an input pin only. SW Reset or WDT Reset have no effect on this pin '1': $\overline{\text{RSTIN}}$ pin is a bidirectional pin. This pin is pulled low during 1024 TCL during reset sequence.
OWDDIS not allocated in ST10F167	<b>Oscillator Watchdog Disable Control</b> '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 $\mu\text{s}$ , 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 not allocated in ST10F167	<b>Power Down Mode Configuration Control</b> '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 occur 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.
CSCFG not allocated in ST10F167	<b>Chip Select Configuration Control</b> '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	<b>Write Configuration Control</b> (Inverted copy of WRC bit of RP0H) '0': Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ retain their normal function '1': Pin $\overline{\text{WR}}$ acts as WRL, pin $\overline{\text{BHE}}$ acts as WRH
CLKEN	<b>System Clock Output Enable (CLKOUT)</b> '0': CLKOUT disabled, pin may be used for general purpose I/O '1': CLKOUT enabled, pin outputs the system clock signal
BYTDIS	<b>Disable/Enable Control for Pin BHE</b> (Set according to data bus width) '0': Pin $\overline{\text{BHE}}$ enabled '1': Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose I/O

Bit	Function
ROMEN	<b>Internal Memory Enable</b> (Set according to pin $\overline{EA}$ during reset) '0': Internal ROM memory disabled, accesses to the ROM memory area use the external bus '1': Internal ROM memory enabled, access to the internal ROM memory (see Table 45 on page 276)
SGTDIS	<b>Segmentation Disable/Enable Control</b> '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit) '1': Segmentation disabled (Only IP is saved/restored)
ROMS1	<b>Internal Memory Mapping</b> '0': Internal ROM memory area mapped to segment 0 (00'0000h...00'7FFFh) '1': Internal ROM memory area mapped to segment 1 (01'0000h...01'7FFFh)
STKSZ	<b>System Stack Size</b> 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. Bit 13 is not available to the ST10F167.

**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		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	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	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	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	RW			RW

## ST10X167

Bit	Function
MCTC	<b>Memory Cycle Time Control</b> (Number of memory cycle time waitstates) 0 0 0 0: 15 waitstates (Number of waitstates = 15 - [MCTC]) ... 1 1 1 1: No waitstates
RWDCx	<b>Read/Write Delay Control for BUSCONx</b> '0': With read/write delay, the CPU inserts 1 TCL after falling edge of ALE '1': No read/write delay, RW is activated after falling edge of ALE
MTTCx	<b>Memory Tristate Time Control</b> '0': 1 waitstate '1': No waitstate
BTYP	<b>External Bus Configuration</b> 0 0: 8 Bit Demultiplexed Bus 0 1: 8 Bit Multiplexed Bus 1 0: 16 Bit Demultiplexed Bus 1 1: 16 Bit Multiplexed Bus <b>Note:</b> For BUSCON0 BTYP is defined via PORT0 during reset.
ALECTLx	<b>ALE Lengthening Control</b> '0': Normal ALE signal '1': Lengthened ALE signal
BUSACTx	<b>Bus Active Control</b> '0': External bus disabled '1': External bus enabled (within the respective address window, see ADDRSEL)
RDYENx	<b>READY Input Enable</b> '0': External bus cycle is controlled by Bit field MCTC only '1': External bus cycle is controlled by the READY input signal
RDYPOLx not allocated in ST10F167	<b>Ready Active Level Control</b> '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	<b>Read Chip Select Enable</b> '0': The $\overline{CS}$ signal is independent of the read command ( $\overline{RD}$ ) '1': The CS signal is generated for the duration of the read command
CSWENx	<b>Write Chip Select Enable</b> '0': The $\overline{CS}$ signal is independent of the write command ( $\overline{WR}, \overline{WRL}, \overline{WRH}$ ) '1': The 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	<b>Range Size Selection</b> Defines the size of the address area controlled by the respective BUSCONx/ADDRSELx register pair. See Table 19.
RGSAD	<b>Range Start Address</b> Defines the upper Bit of the start address (A23...) of the respective address area. See Table 19.

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.

**8.4.1 - 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 ST10X167. 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 19).

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 19** : Definition of address areas

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

**8.4.2 - Address Window Arbitration**

**This feature does not exist for the ST10F167.**

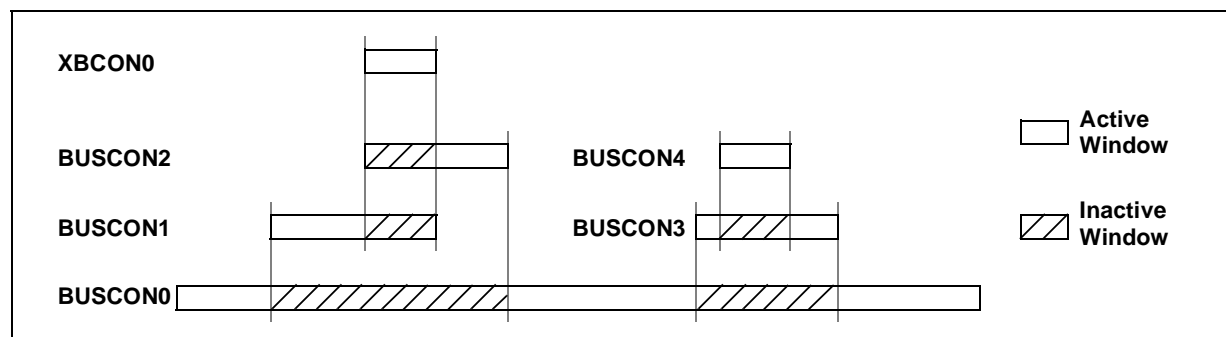
For each access the EBC compares the current address with all address select registers (programmable ADDRSELx and hardwired XADRSx). This comparison is done in four levels.

- 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 59).

- 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 59** : 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				ResetValue:--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 <sup>1</sup>	<b>Write Configuration Control</b> (Set according to pin POH.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 <sup>1</sup>	<b>Chip Select Line Selection</b> (Number of active CS outputs) 0 0: 3 $\overline{CS}$ lines: $\overline{CS2}...$ $\overline{CS0}$ 0 1: 2 $\overline{CS}$ lines: $\overline{CS1}...$ $\overline{CS0}$ 1 0: No $\overline{CS}$ lines at all 1 1: 5 $\overline{CS}$ lines: $\overline{CS4}...$ $\overline{CS0}$ (Default without pull-downs)		
SALSEL	<b>Segment Address Line Selection</b> (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	POH.7-5	CPU Frequency $f_{CPU} = f_{XTAL} \times F$	Notes
	1 1 1	$f_{XTAL} \times 4$	$f_{XTAL} \times 4$ Default configuration <sup>1</sup>
	1 1 0	$f_{XTAL} \times 3$	$f_{XTAL} \times 4$
	1 0 1	$f_{XTAL} \times 2$	$f_{XTAL} \times 4$
	1 0 0	$f_{XTAL} \times 5$	$f_{XTAL} \times 4$
	0 1 1	$f_{XTAL} \times 1$	$f_{XTAL} \times 1$ Direct drive
	0 1 0	$f_{XTAL} \times 1.5$	$f_{XTAL} \times 1$
	0 0 1	$f_{XTAL} \times 0.5$	$f_{XTAL} \times 1$ CPU clock via prescaler <sup>2</sup>
	0 0 0	$f_{XTAL} \times 2.5$	$f_{XTAL} \times 1$

Notes 1. In ST10X167, RPOH.[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.

### 8.4.3 - 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 demultiplexed external bus, even for multiplexed bus cycles.
- Not all address areas defined via registers ADDRSELx may overlap each other. The operation of the EBC will be unpredictable in such a case. See Section 8.4.2 - 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).

### 8.5 - 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 20).

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 20). The address mentioned above includes Port1, Port 4,  $\overline{BHE}$  and ALE which also pulses for an XBUS cycle. The external  $\overline{CS}$  signals on Port 6 are driven inactive (high) because the EBC switches to an internal XCS signal.

The **external control signals** ( $\overline{RD}$  and  $\overline{WR}$  or  $\overline{WRL}/\overline{WRH}$  if enabled) **remain inactive** (high) (see Table 20).

**Table 20** : Status of the external bus interface during EBC idle state

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

**8.6 - 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 ST10X167 supports this approach with the possibility to arBtrate the access to its external bus, and to the external devices.

This bus arbitration allows an external master to request the ST10X167's bus via the  $\overline{HOLD}$  input. The ST10X167 acknowledges this request via the  $\overline{HLDA}$  output and will float its bus lines in this case. The  $\overline{CS}$  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 ST10X167. During this time the ST10X167 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 ST10X167 needs access to its external bus while it is occupied by another bus master, it demands it via the  $\overline{BREQ}$  output.

The external bus arbitration is enabled by setting Bit  $\overline{HLDEN}$  in register  $\overline{PSW}$  to '1'. In this case the three bus arbitration pins  $\overline{HOLD}$ ,  $\overline{HLDA}$  and  $\overline{BREQ}$  are automatically controlled by the EBC independent of their I/O configuration. Bit  $\overline{HLDEN}$  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 ST10X167 will not answer to  $\overline{HOLD}$  requests from other external masters. If  $\overline{HLDEN}$  is cleared while the ST10X167 is in hold state (code execution from internal RAM/ROM) this hold state is left only after  $\overline{HOLD}$  has been deactivated again. In this case the current hold state continues and only the next  $\overline{HOLD}$  request is not answered.

Connecting two ST10X167's in this way would require additional logic to combine the respective output signals  $\overline{HLDA}$  and  $\overline{BREQ}$ . This can be avoided by switching one of the controllers into slave mode where pin  $\overline{HLDA}$  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  $\overline{DP6.7}$  to '1'.  $\overline{DP6.7} = '0'$  (default after reset) selects the Master Mode.

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

### 8.6.1 - Connecting Bus Masters

When multiple ST10X167's or a ST10X167 and another bus master shall share external resources some glue logic is required that defines the currently active bus master and also enables a ST10X167 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 ST10X167 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 ST10X167 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 60).

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.

### 8.6.2 - Entering the Hold State

Access to the ST10X167's external bus is requested by driving its HOLD input low. After synchronizing this signal the ST10X167 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 ST10X167 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 ST10X167 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 61).

**Figure 60** : Sharing external resources using slave mode

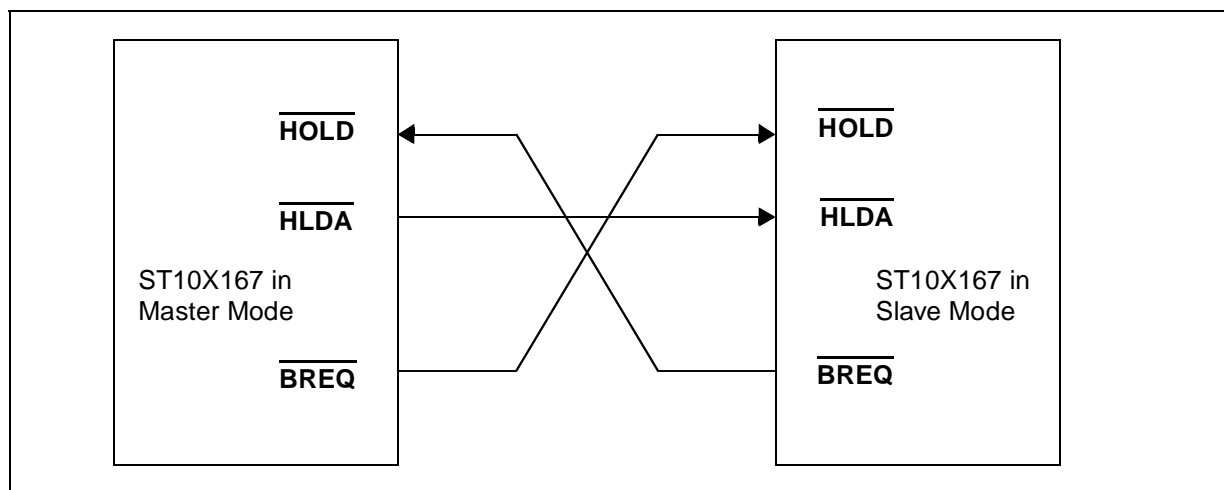
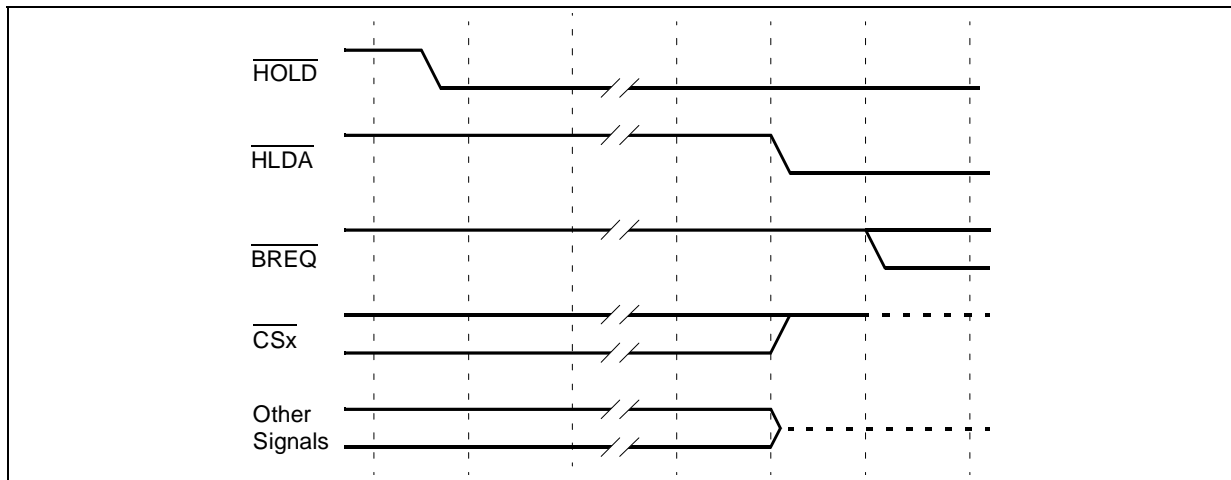


Figure 61 : External bus arbitration, releasing the bus



Note The ST10X167 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.

control signals and resume executing external bus cycles if required. Depending on the arbitration logic, the external bus can be returned to the ST10X167 under two circumstances:

- The external master does no more require access to the shared resources and gives up its own access rights.
- The ST10X167 needs access to the shared resources and demands this by activating its BREQ output.

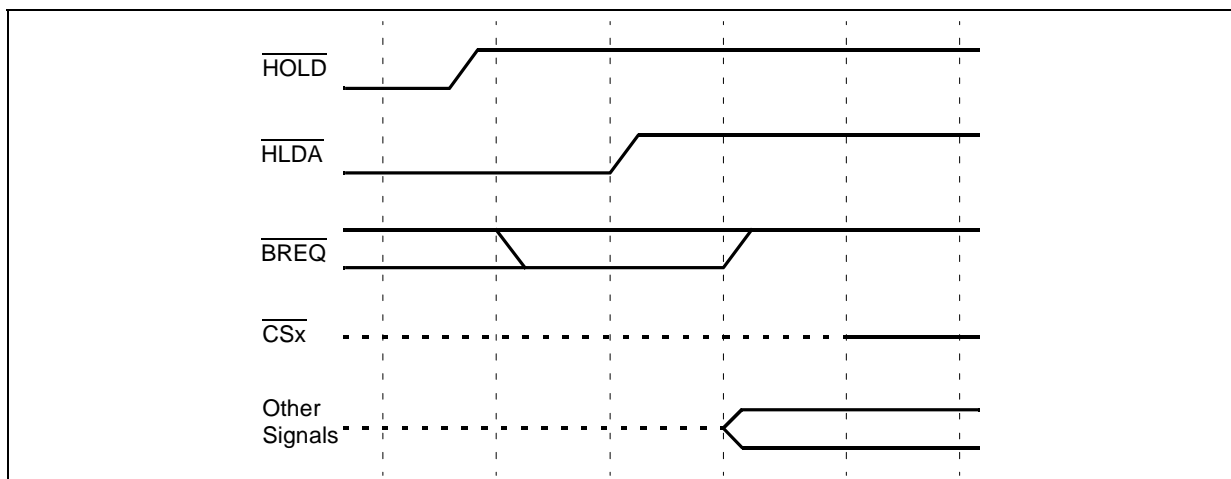
The arbitration logic may then deactivate the other master's HLDA and so free the external bus for the ST10X167, depending on the priority of the different masters.

Note The Hold State is not terminated by clearing Bit HLDEN.

### 8.6.3 - Exiting the Hold State

The external bus master returns the access rights to the ST10X167 by driving the HOLD input high. After synchronizing this signal the ST10X167 will drive the HLDA output high, actively drive the

Figure 62 : 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  $\text{HOLD}$  going high.  $\overline{\text{BREQ}}$  and  $\text{HOLD}$  are connected via an external arbitration circuitry. Please note that  $\overline{\text{HOLD}}$  may also be deactivated without the ST10X167 requesting the bus.

### 8.7 - The XBUS Interface

The ST10X167 provides an on-chip interface (the XBUS interface), which allows to connect integrated customer / application specific peripherals to the standard controller core.

The XBUS is an internal representation of the external bus interface, 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.

Interrupt nodes and configuration pins are provided for X-Peripherals to be integrated.

## 9 - 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.

### 9.1 - 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 63.

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 Bitaddressable SFR space. When any of the timer registers is written to by the CPU in the state immediately before a timer increment, decrement, reload, or capture, the CPU write operation has priority. This is to guarantee correct results.



Figure 63 : SFRs and port pins associated with timer block GPT1

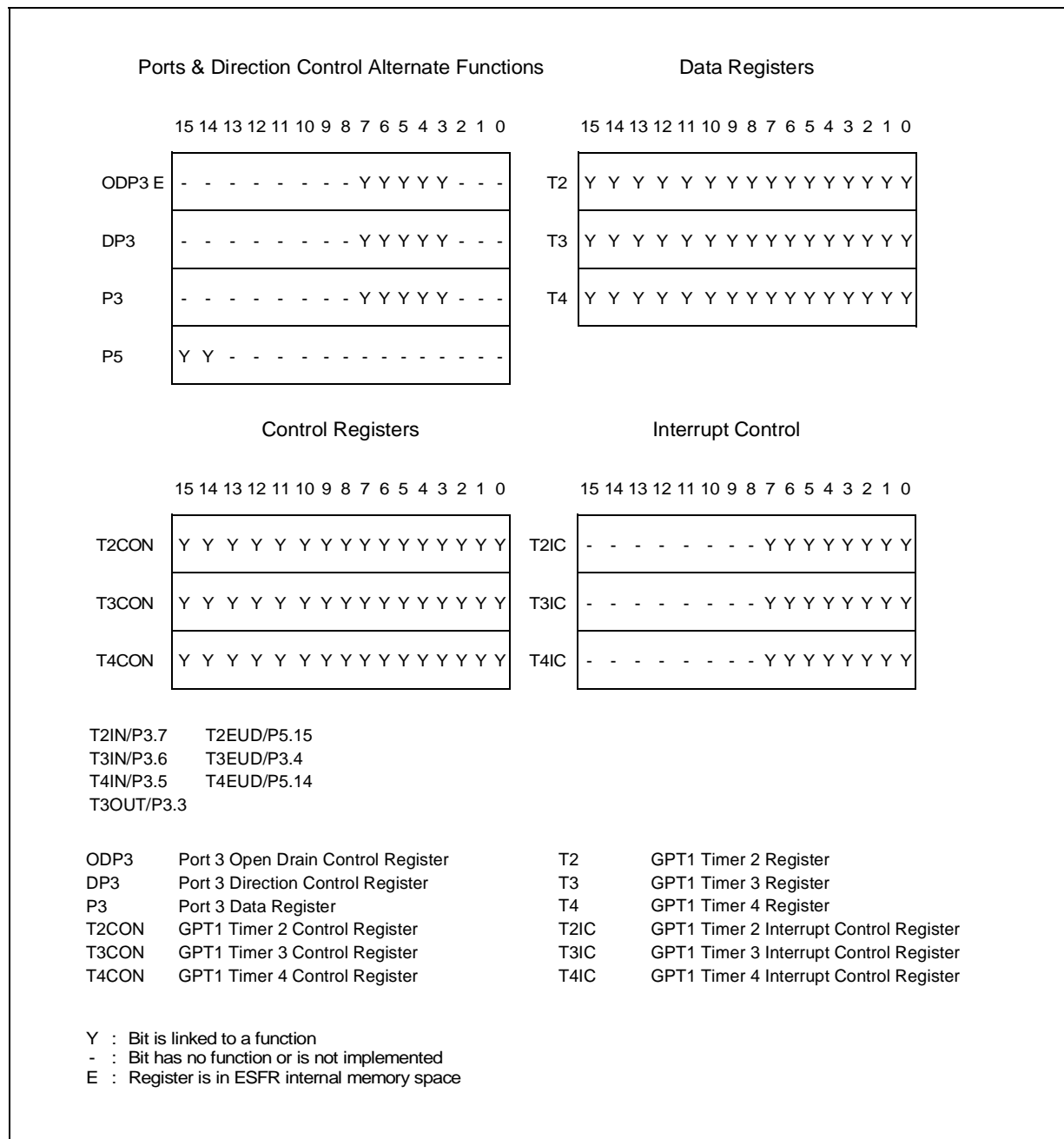
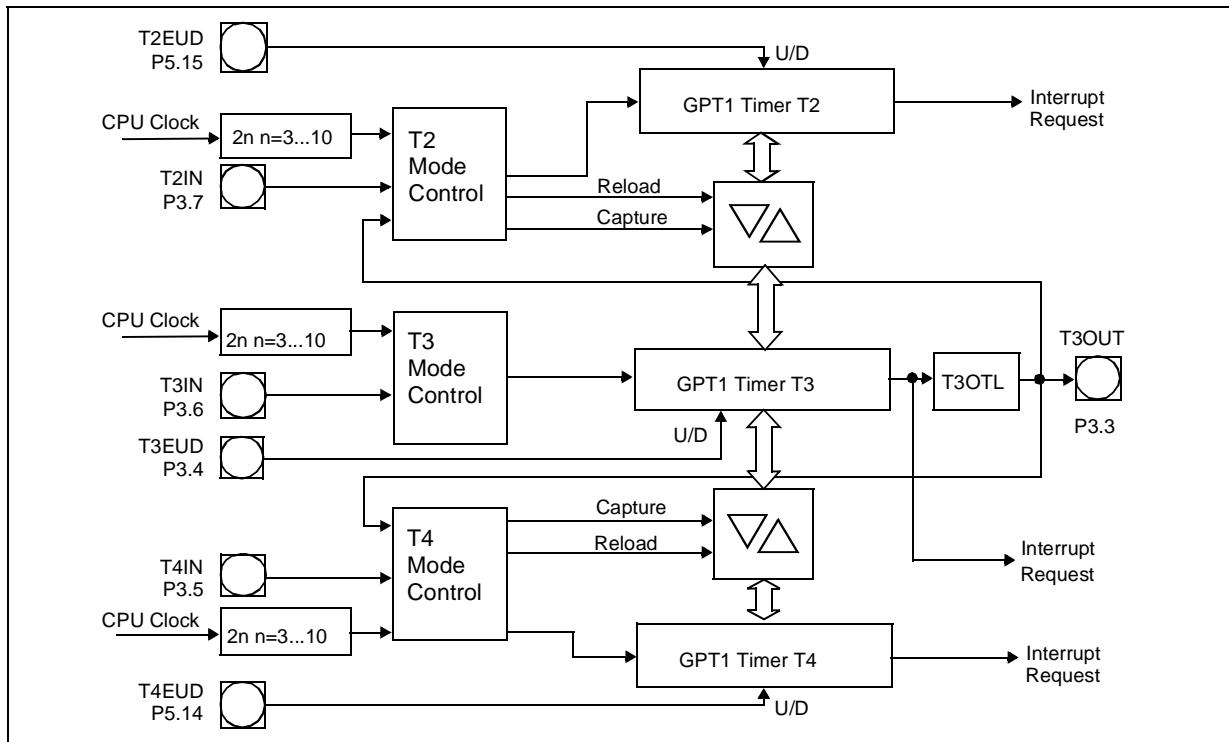


Figure 64 : GPT1 block diagram



9.1.1 - GPT1 Core Timer T3

The core timer T3 is configured and controlled via its Bitaddressable 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	<b>Timer 3 Input Selection</b> - Depends on the operating mode, see respective sections.
T3M	<b>Timer 3 Mode Control</b> (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 (not for ST10F167) 1 1 1: Reserved Do not use this combination.
T3R	<b>Timer 3 Run Bit</b> : T3R = '0':Timer / Counter 3 stops - T3R = '1':Timer / Counter 3 runs
T3UD	<b>Timer 3 Up / Down Control</b> <sup>1</sup>
T3UDE	<b>Timer 3 External Up/Down Enable</b> <sup>1</sup>
T3OE	<b>Alternate Output Function Enable</b> T3OE = '0': Alternate Output Function Disabled - T3OE = '1':Alternate Output Function Enabled
T3OTL	<b>Timer 3 Output Toggle Latch</b> - 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 21.

### 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 21.

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 21** : 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 Bit 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

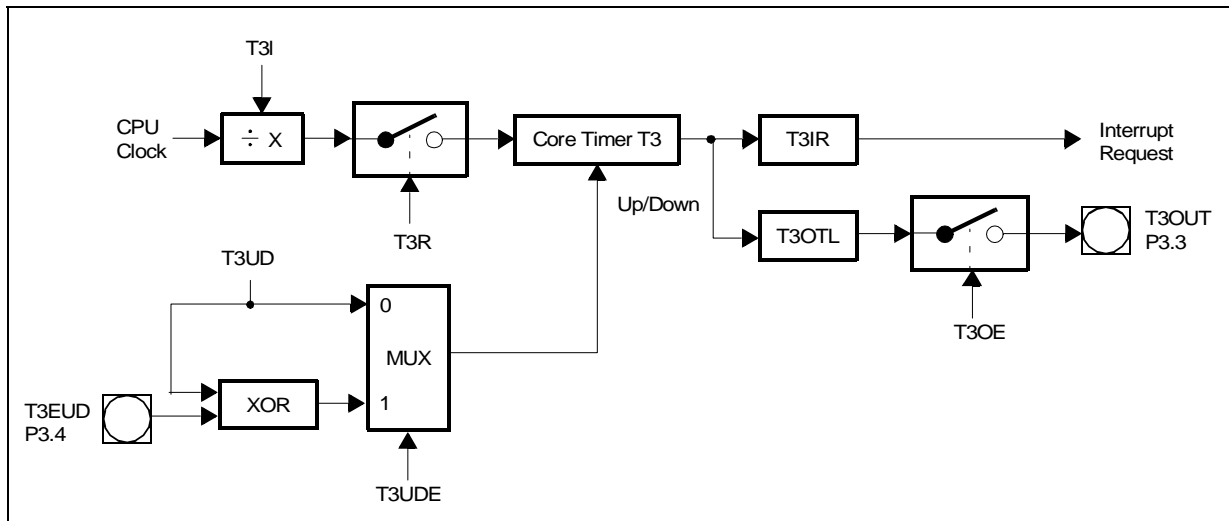
Timer mode for the core timer T3 is selected by setting Bit field T3M in register T3CON to '000b'. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable pre-scaler, which is selected by Bit field T3I.

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:

$$f_{T3} = \frac{f_{CPU}}{8 \times 2^{(T3I)}}$$

$$r_{T3} [\mu s] = \frac{8 \times 2^{(T3I)}}{f_{CPU} [MHz]}$$

Figure 65 : Core timer T3 in timer mode



The timer resolutions which result from the selected pre-scaler option are listed in the Table 22. 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 22 : 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 66). 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 23).

Figure 66 : Core timer T3 in gated timer mode

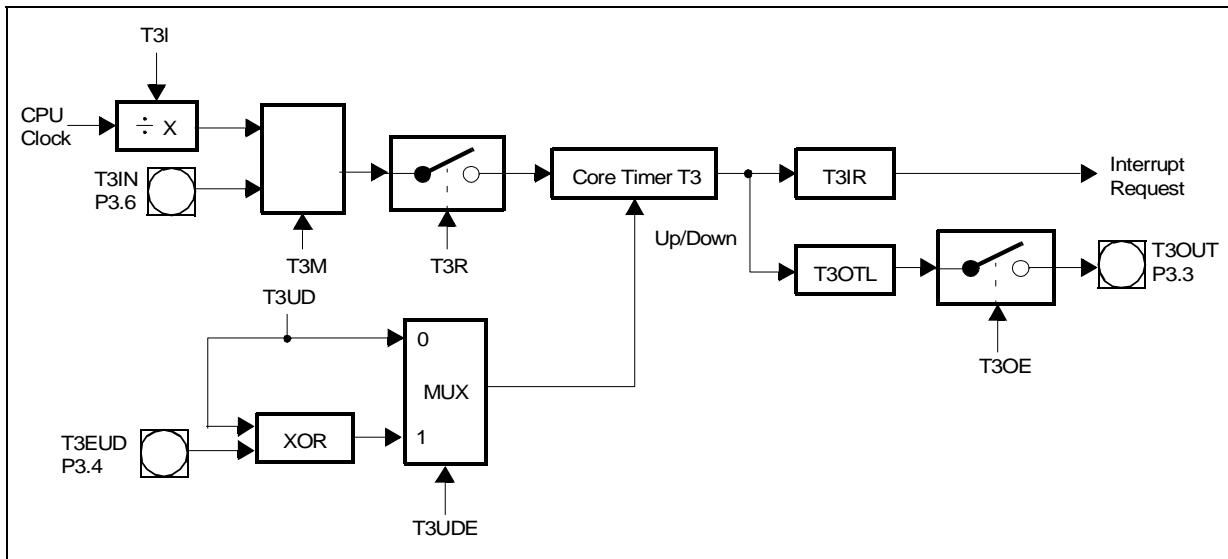


Figure 67 : Core timer T3 in counter mode

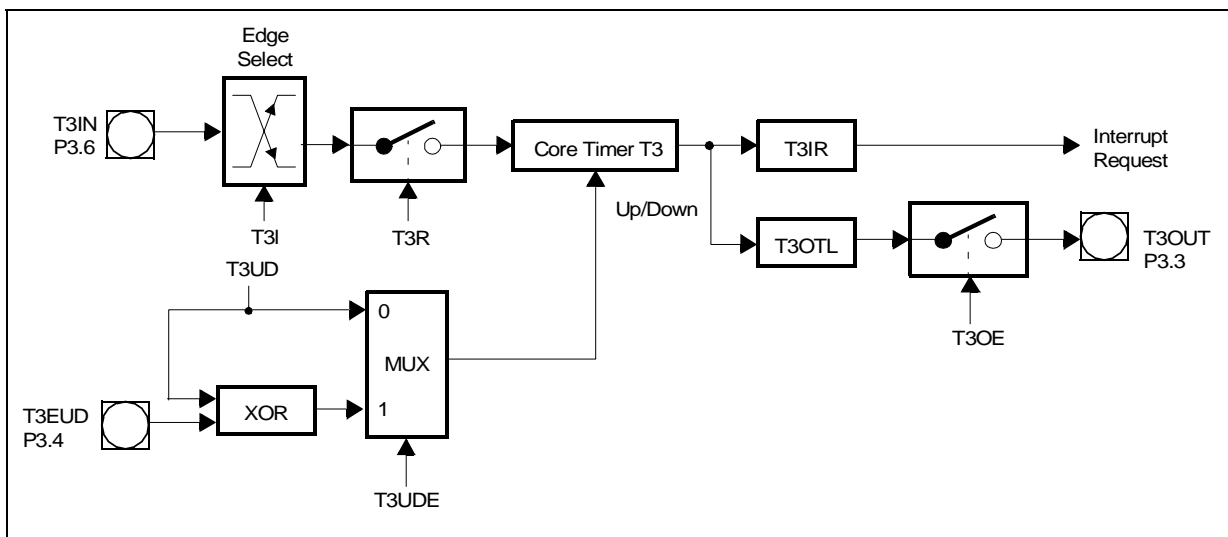


Table 23 : GPT1 core timer T3 (counter mode) input edge selection

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**

Note: This function is not available for ST10F167.

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 68).

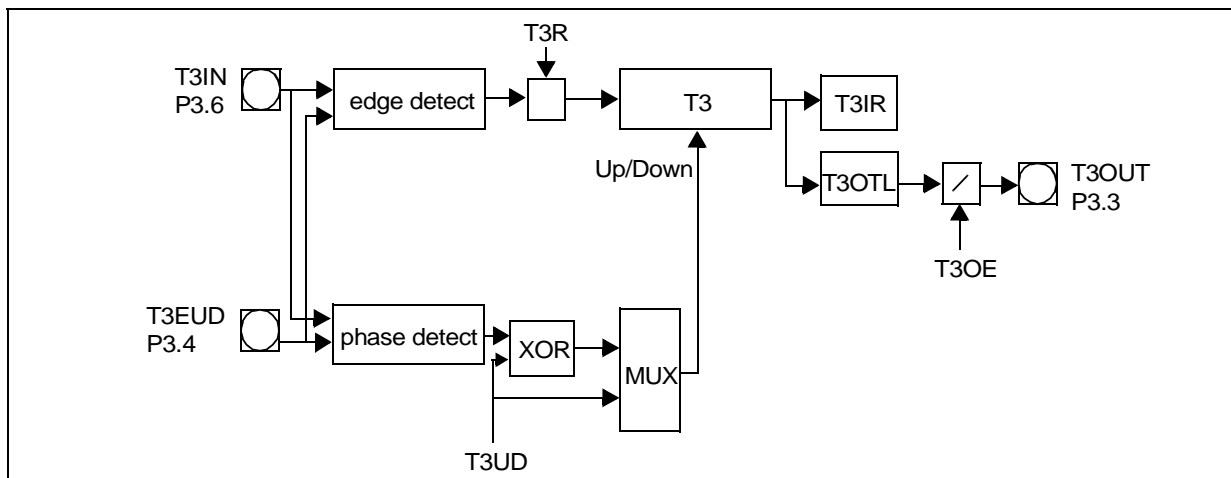
Bitfield T3I in control register T3CON selects the triggering transitions (see Table 24). In this mode the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal.

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  $\bar{A}$ ) to digital signals (as A) digital signals. this greatly increases noise immunity.

The third encoder output "Top0" which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset timer T3 (for example, via PEC transfer from ZEROS) (see Figure 69).

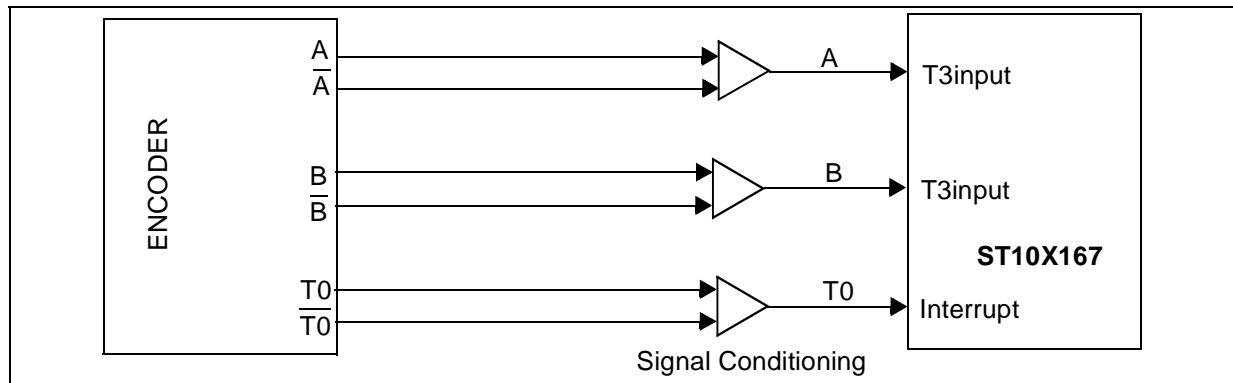
**Figure 68** : Core timer T3 in incremental interface mode



**Table 24** : 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 69 : Connection of the encoder to the ST10X167



For incremental interface operation the following conditions must be met

- Bitfield 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.

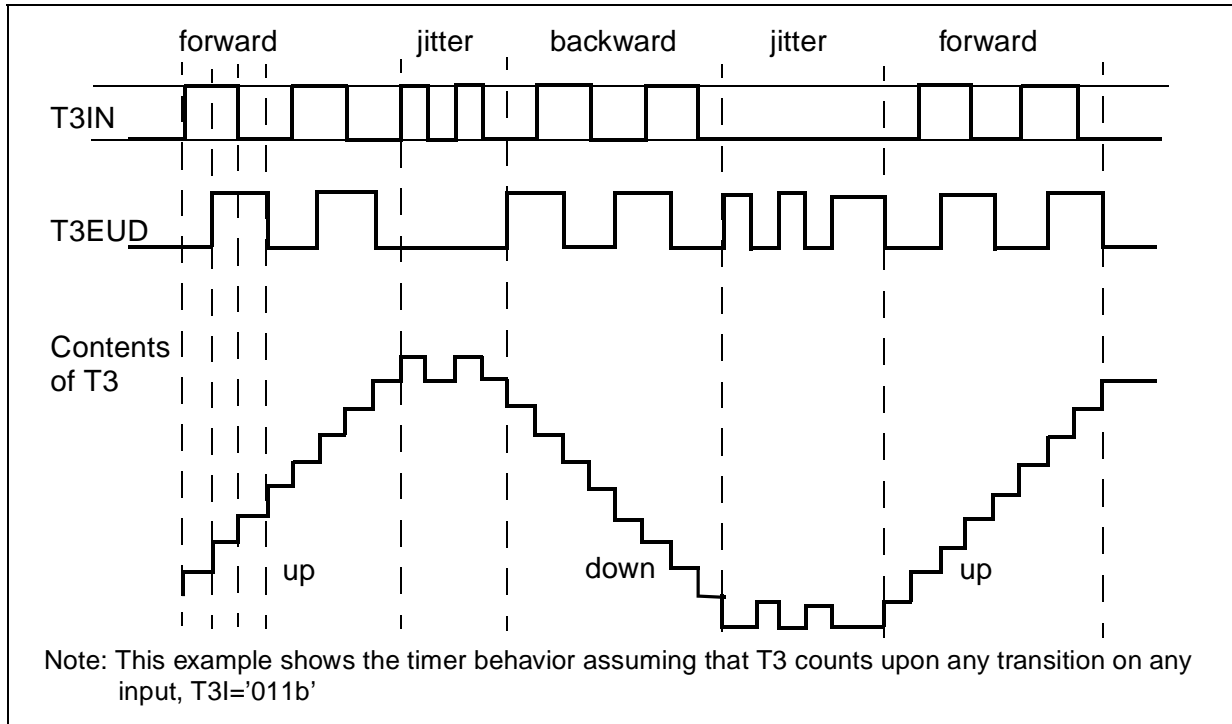
In incremental interface mode, the count direction is automatically derived from the sequence in which the input signals change.

This corresponds to the rotation direction of the connected sensor. The table below summarizes the possible combinations.

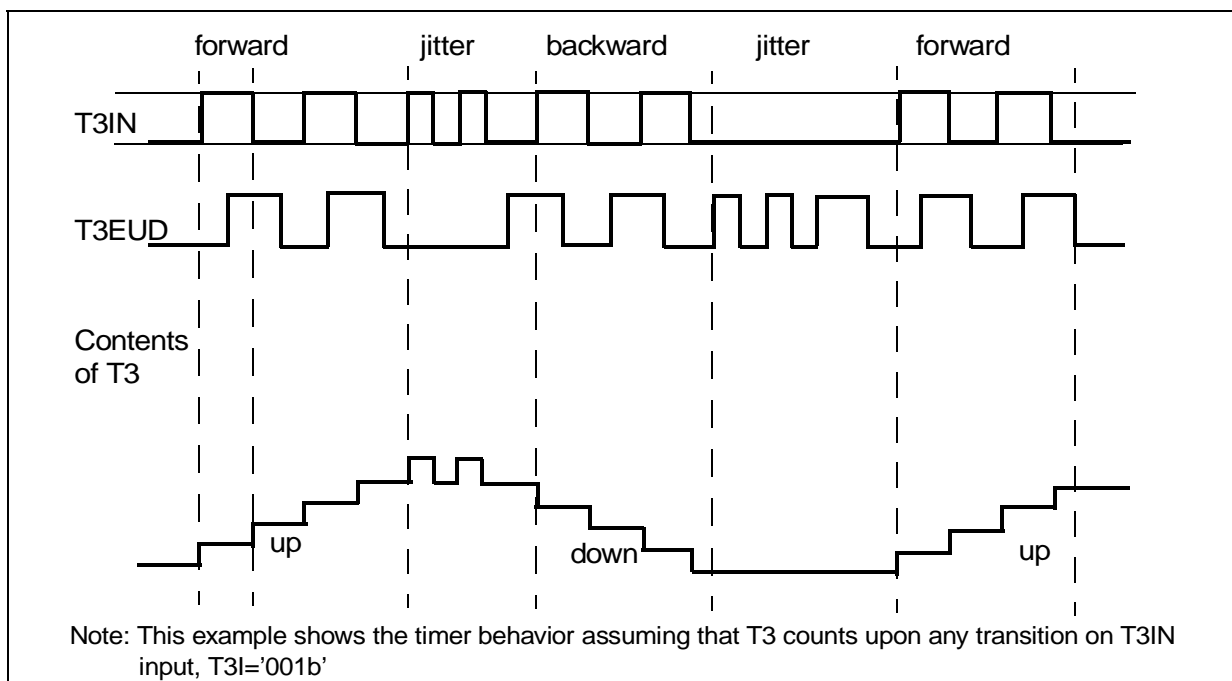
Level on respective other input	T3IN Input		T3EUD Input	
	Rising	Falling	Rising	Falling
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

The Figure 70 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 70** : Evaluation of the incremental encoder signals



**Figure 71** : Evaluation of the incremental encoder signals





Note Timer 3 operating in incremental interface mode automatically provides information on the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods. This is facilitated by an additional special capture mode for timer T5.

### 9.1.2 - 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 Bitaddressable 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	<b>Timer x Input Selection</b> Depends on the Operating Mode, see respective sections.
TxM	<b>Timer x Mode Control (Basic Operating Mode)</b> 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 (not for ST10F167) 1 1 X: Reserved. Do not use this combination
TxR	<b>Timer x Run Bit</b> TxR = '0': Timer / Counter x stops TxR = '1': Timer / Counter x runs
TxUD	<b>Timer x Up / Down Control</b> <sup>1</sup>
TxUDE	<b>Timer x External Up/Down Enable</b> <sup>1</sup>

Note 1. For the effects of Bit TxUD and TxUDE refer to the direction Table 24 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

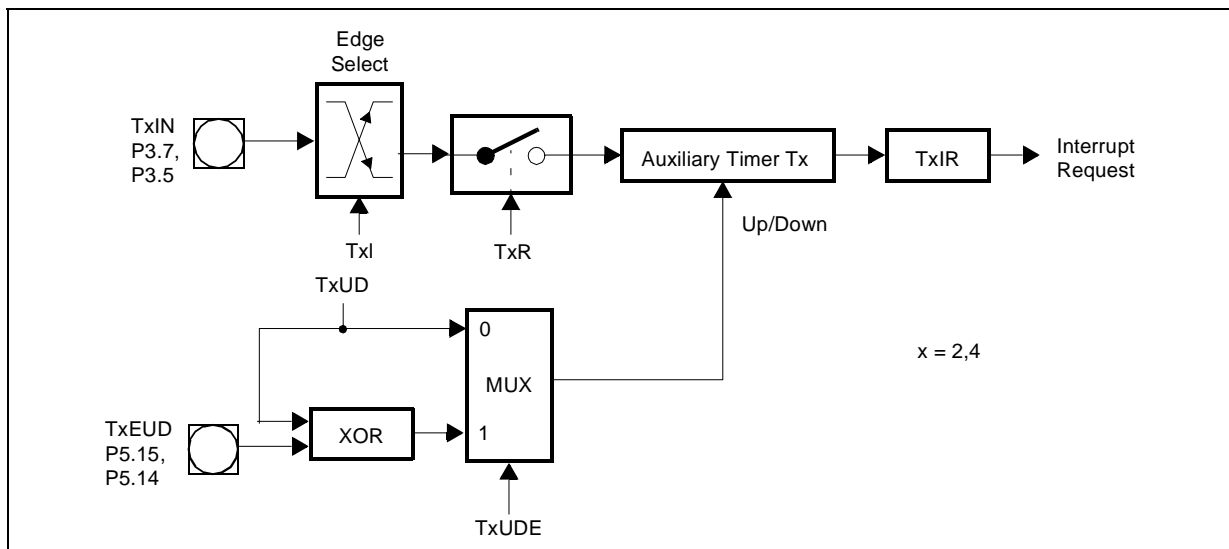
transition of timer T3's output toggle latch T3OTL (see Figure 72).

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 25).

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 72** : Auxiliary timer in counter mode



**Table 25** : 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.

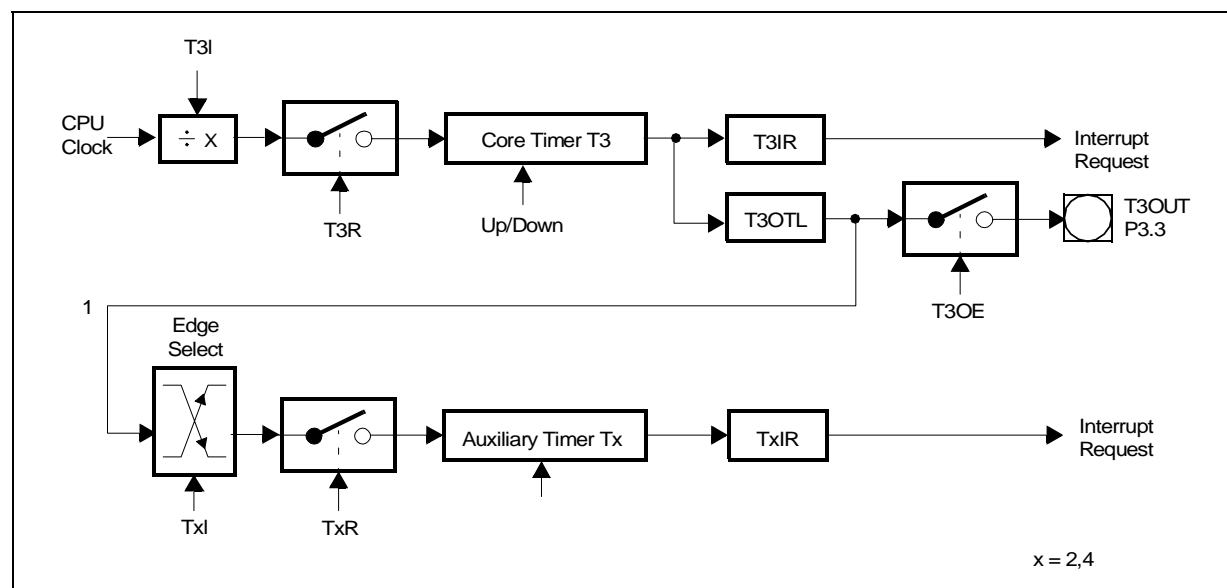
T3 can operate in timer, gated timer or counter mode in this case (see Figure 73).

### 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 25). A transition of the auxiliary timer's input or the output toggle latch T3OTL may trigger the reload.

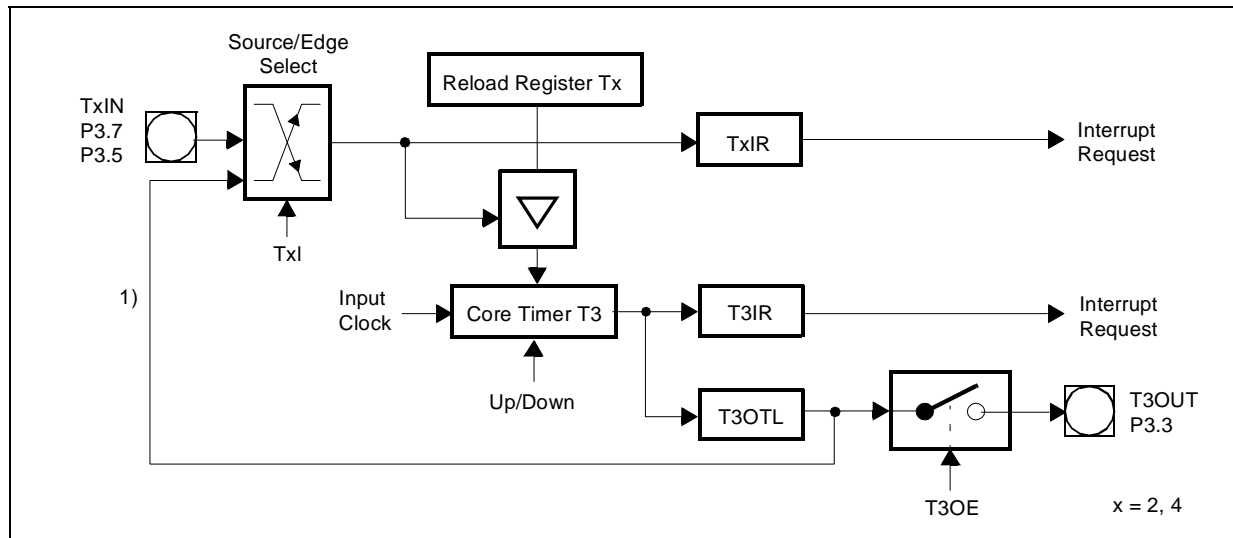
**Note** When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.

**Figure 73 :** 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 74 : 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 75 shows an example for the generation of a PWM signal using the alternate reload mechanism.

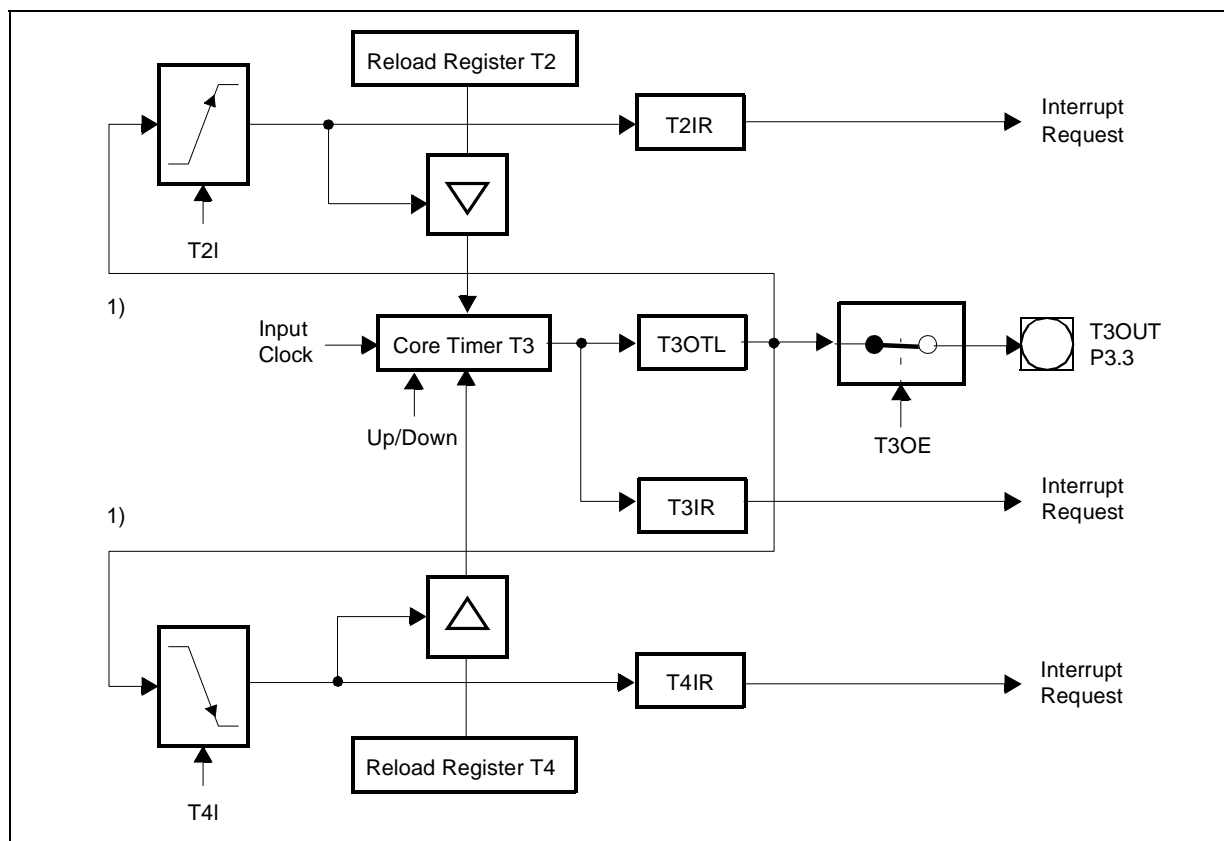
T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions).

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 75 : 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'.

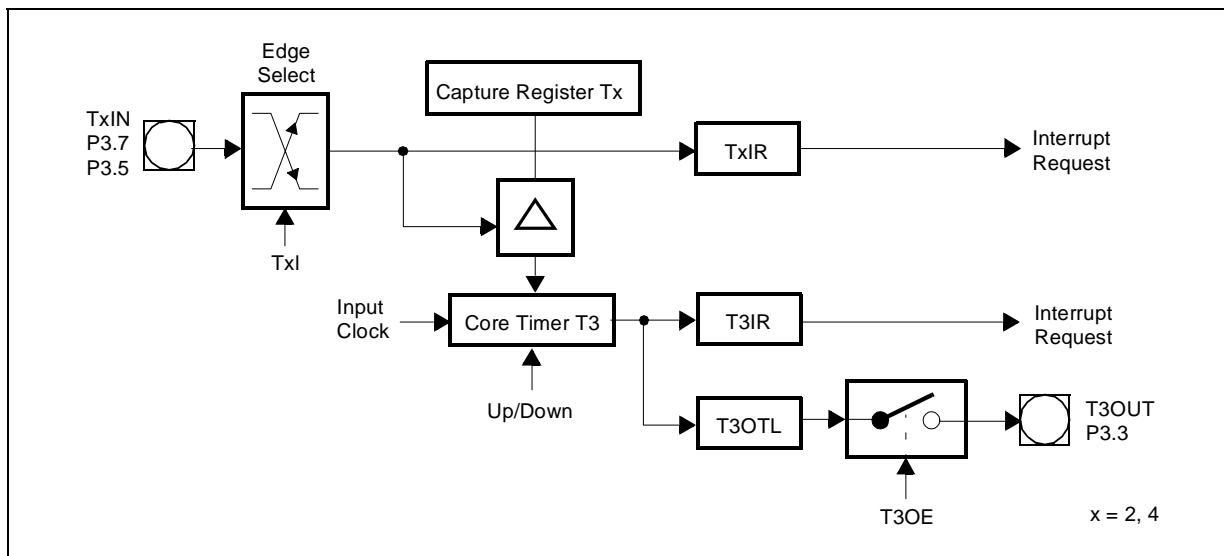
In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN.

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 76 : 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.

9.1.3 - 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.

## 9.2 - 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 77.

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 Bitaddressable SFRs T5 and T6.

Figure 77 : SFRs and port pins associated with timer block GPT2

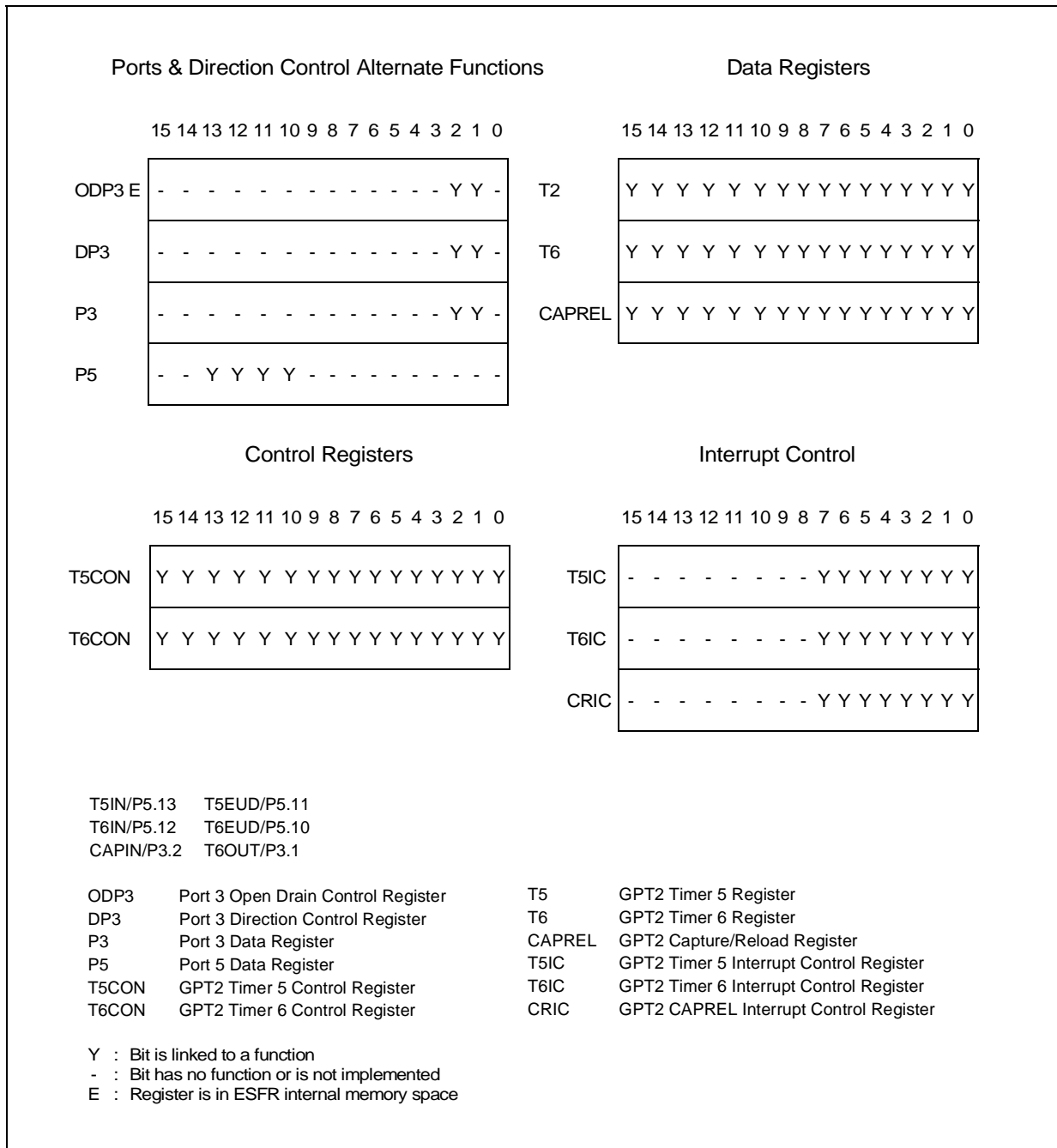
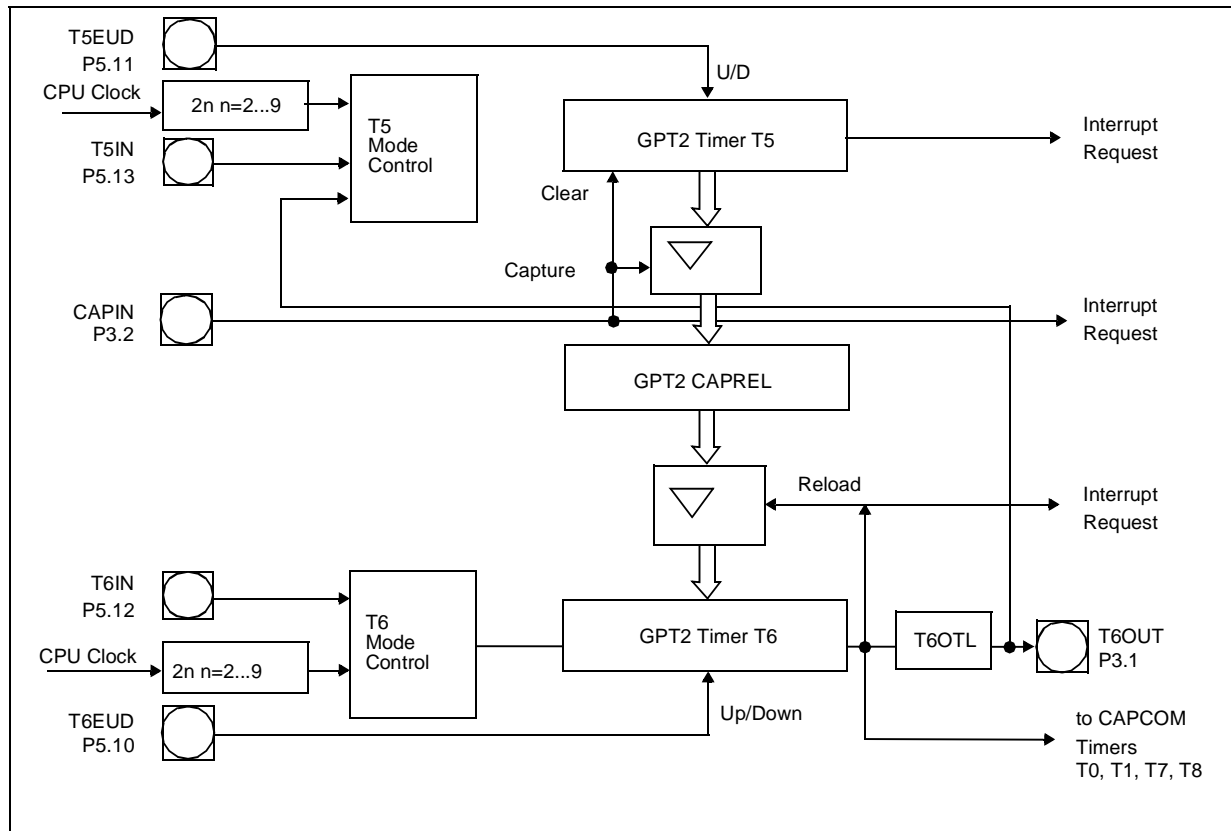




Figure 78 : GPT2 block diagram



### 9.2.1 - GPT2 Core Timer T6

The operation of the core timer T6 is controlled by its Bitaddressable 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	<b>Timer 6 Input Selection</b> Depends on the Operating Mode, see respective sections.
T6M	<b>Timer 6 Mode Control (Basic Operating Mode)</b> 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	<b>Timer 6 Run Bit</b> T6R = '0': Timer / Counter 6 stops T6R = '1': Timer / Counter 6 runs
T6UD	<b>Timer 6 Up / Down Control</b> <sup>1</sup>
T6UDE	<b>Timer 6 External Up/Down Enable</b> <sup>1</sup>
T6OE	<b>Alternate Output Function Enable</b> T6OE = '0': Alternate Output Function Disabled T6OE = '1': Alternate Output Function Enabled
T6OTL	<b>Timer 6 Output Toggle Latch</b> Toggles on each overflow / underflow of T6. Can be set or reset by software.
T6SR	<b>Timer 6 Reload Mode Enable</b> 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 26.

**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 26. 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 26 : 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.

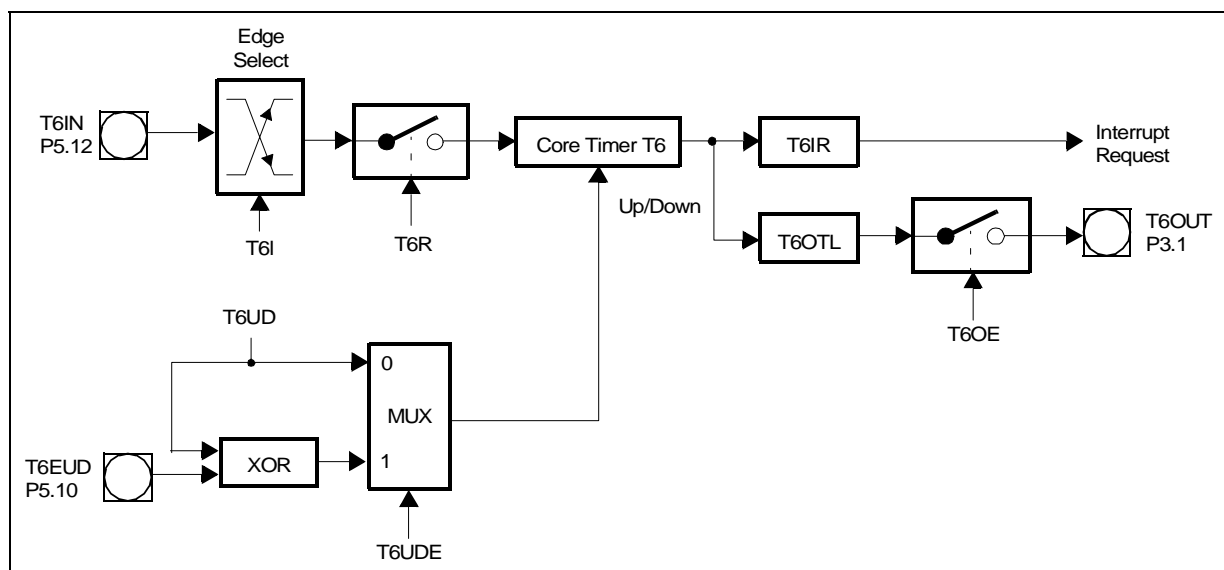
An overflow or underflow of timer T6 can also be used to clock the timers in the CAPCOM units. For this purpose, there is a direct internal connection between timer T6 and the CAPCOM timers.

### 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:

$$f_{T6} = \frac{f_{CPU}}{4 \times 2^{(T6I)}} \quad r_{T6} [\mu s] = \frac{4 \times 2^{(T6I)}}{f_{CPU} [MHz]}$$

**Figure 79** : Block diagram of core timer T6 in timer mode



The timer resolutions which result from the selected pre-scaler option are listed in the Table 27. This table also applies to the Gated Timer Mode of T6 and to the auxiliary timer T5 in timer and gated timer mode.

**Table 27** : 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 80). 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

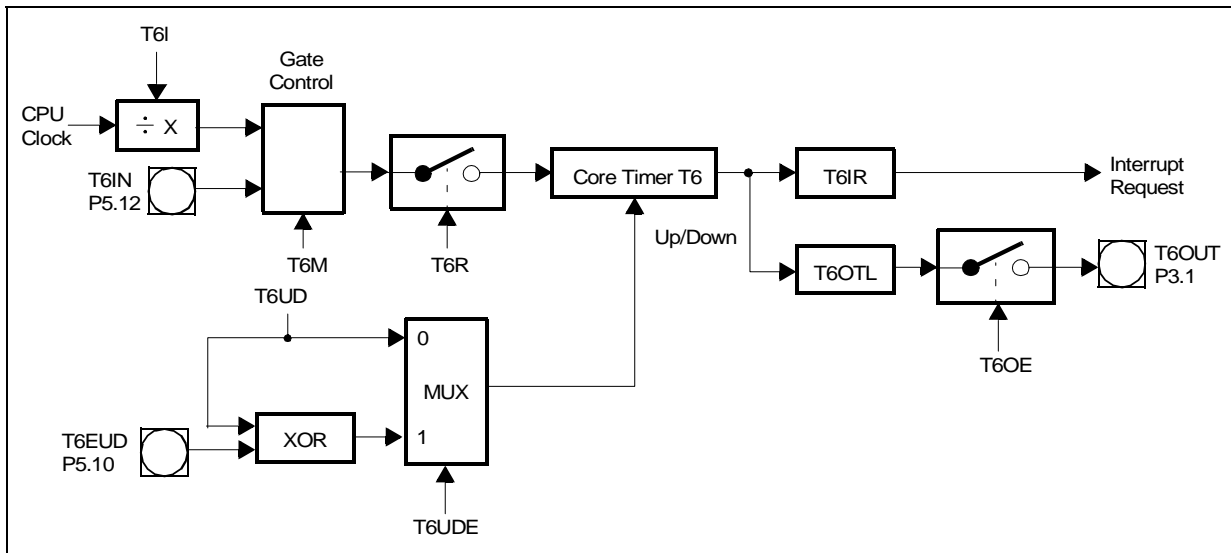
active. It will stop, if either T6R='0' or the gate is inactive.

**Note** A transition of the gate signal at pin T6IN does not cause an interrupt request.

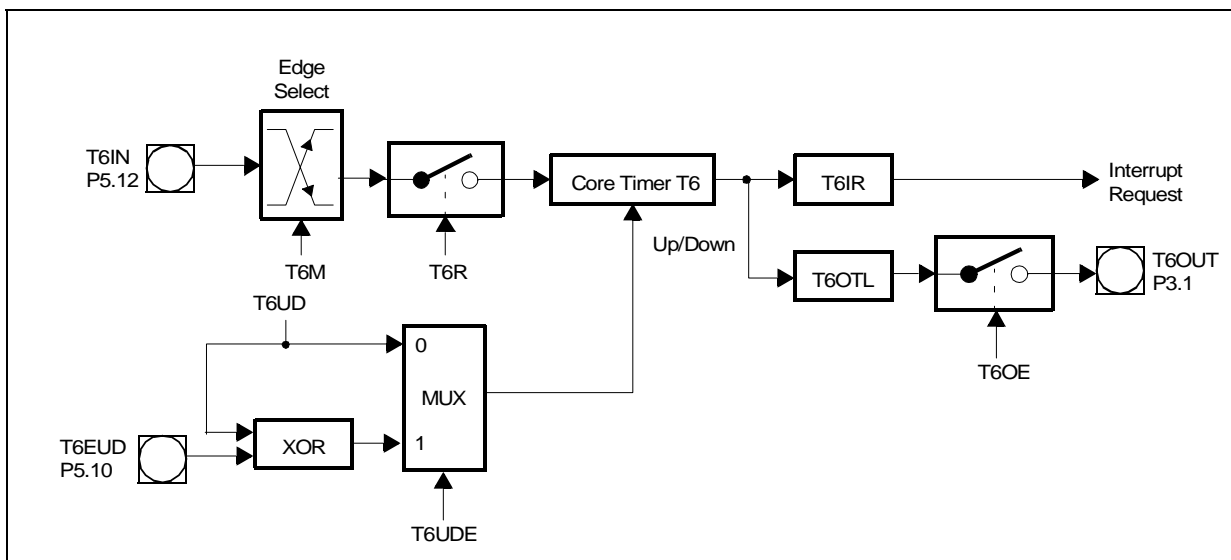
**Timer 6 in Counter Mode**

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 28).

**Figure 80** : Block diagram of core timer T6 in gated timer mode



**Figure 81** : Block diagram of core timer T6 in counter mode



**Table 28** : 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 Bitaddressable control register T5CON. Note that functions which are present in both timers of block GPT2 are controlled in the same Bit positions and in the same manner in each of the specific control registers.

**T5CON (FF46h / A3h)**

SFR

ResetValue: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					

Bit	Function
T5I	<b>Timer 5 Input Selection</b> Depends on the Operating Mode, see respective sections.
T5M	<b>Timer 5 Mode Control (Basic Operating Mode)</b> 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	<b>Timer 5 Run Bit</b> T5R = '0': Timer / Counter 5 stops T5R = '1': Timer / Counter 5 runs
T5UD	<b>Timer 5 Up / Down Control</b> <sup>1</sup>
T5UDE	<b>Timer 5 External Up/Down Enable</b> <sup>1</sup>
CT3	<b>Capture Trigger 3</b> 0: Capture triggered from CAPIN pin 1: Capture triggered from T3 input pin
CI	<b>Register CAPREL Input Selection</b> 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	<b>Timer 5 Clear Bit</b> T5CLR = '0': Timer 5 not cleared on a capture T5CLR = '1': Timer 5 is cleared on a capture
T5SC	<b>Timer 5 Capture Mode Enable</b> 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 26.

**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.

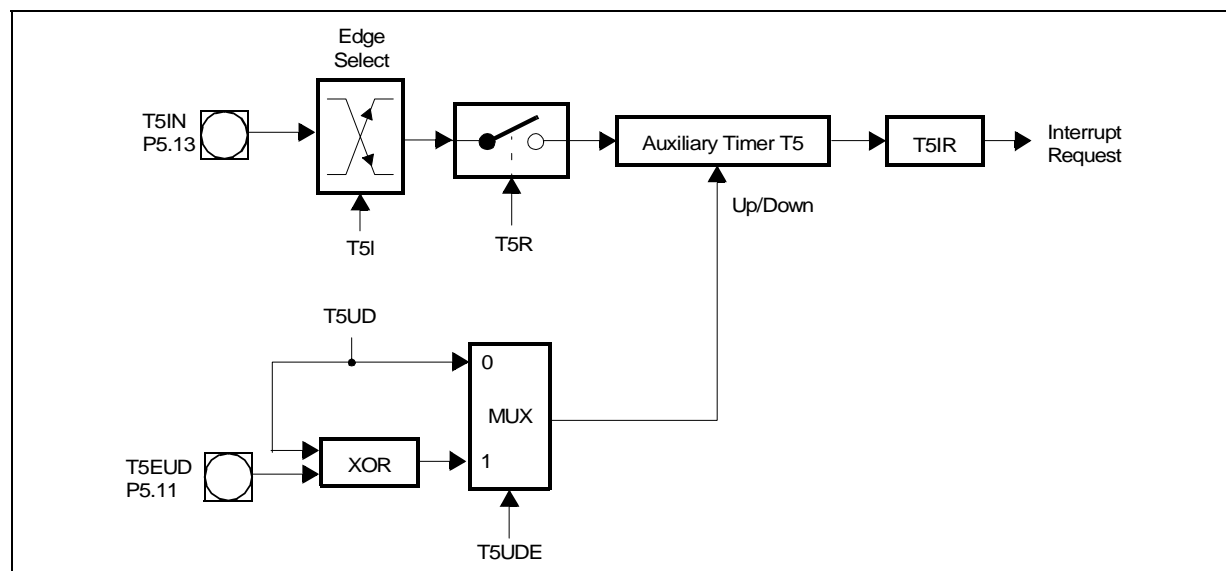
The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at either the input pin, or at the toggle latch T6OTL (see Figure 82).

Bit field T5I in control register T5CON selects the triggering transition (see Table 29).

**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 82** : Block diagram of auxiliary timer T5 in counter mode



**Table 29** : 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 83).

### 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

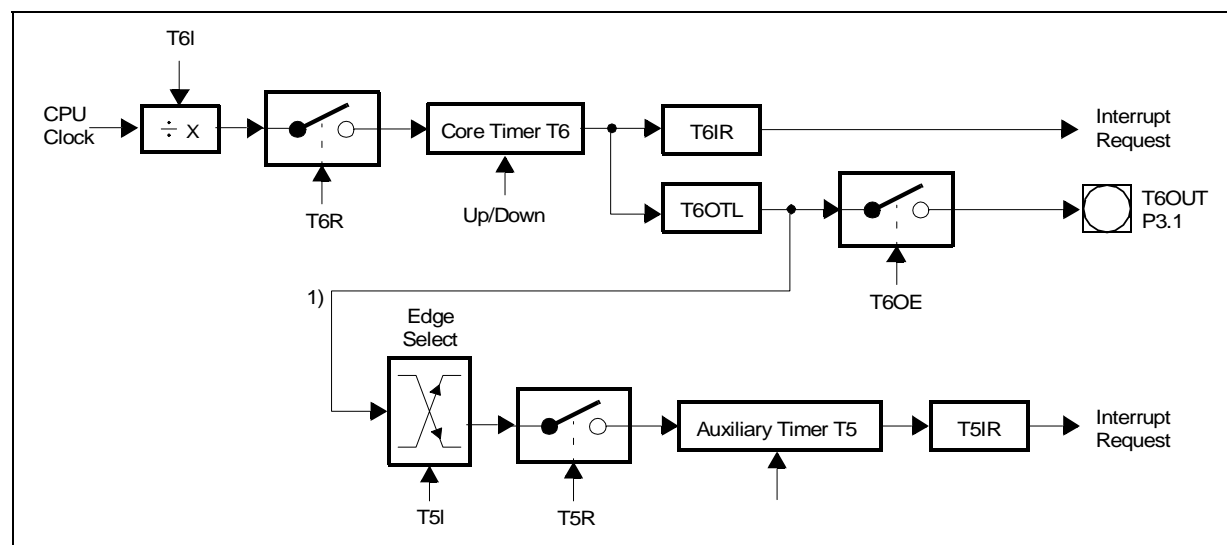
or transitions on input T3IN or input T3EUD or both inputs T3IN and T3EUD. The active edge is controlled by Bit field CI in register T5CON. The maximum input frequency for the capture trigger signal at CAPIN is  $f_{CPU} / 4$ . To ensure that a transition of the capture trigger signal is correctly recognized, 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.

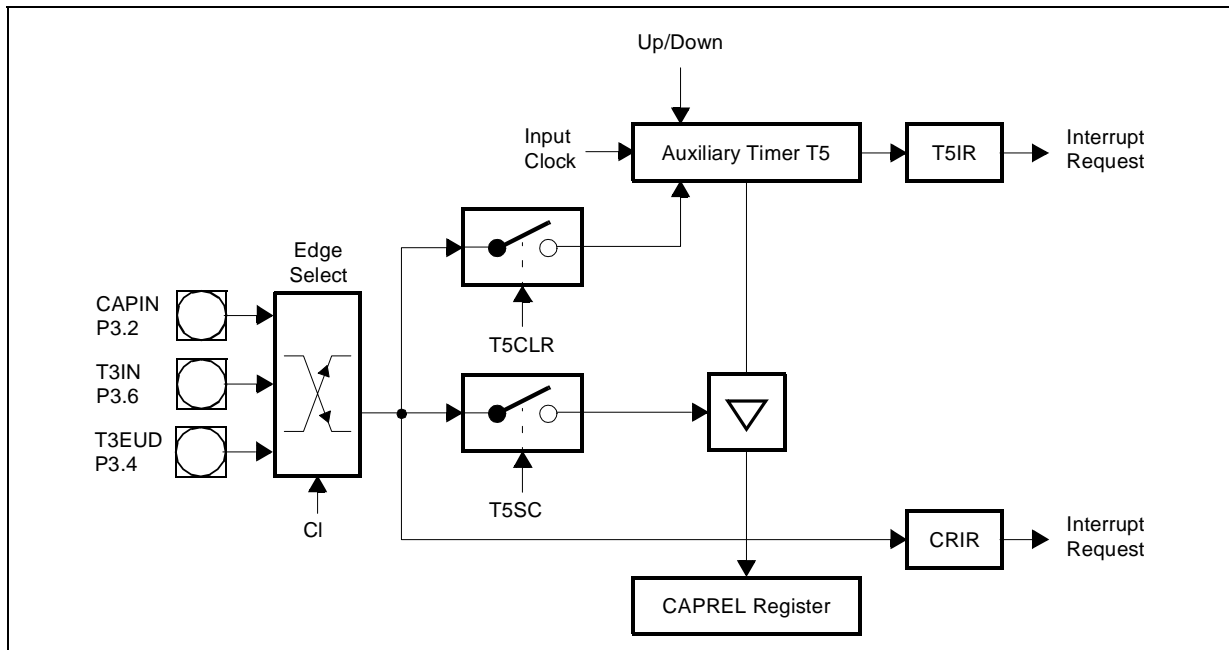
Note Bit T5SC only controls whether a capture is performed or not. If T5SC='0', the input pin CAPIN can still be used to clear timer T5 or as an external interrupt input. This interrupt is controlled by the CAPREL interrupt control register CRIC (see Figure 84).

**Figure 83** : 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 84 : GPT2 register CAPREL in capture mode

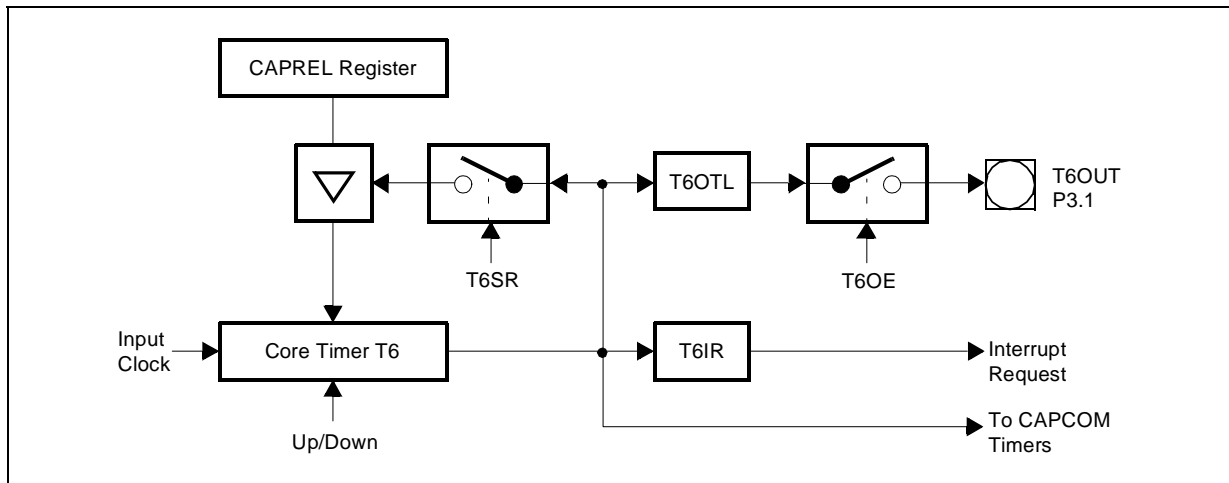


**GPT2 Capture / Reload Register CAPREL in Reload Mode**

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 85 : 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 Bit. This feature can be used to generate an output frequency that is a multiple of the input frequency (see Figure 86).

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.

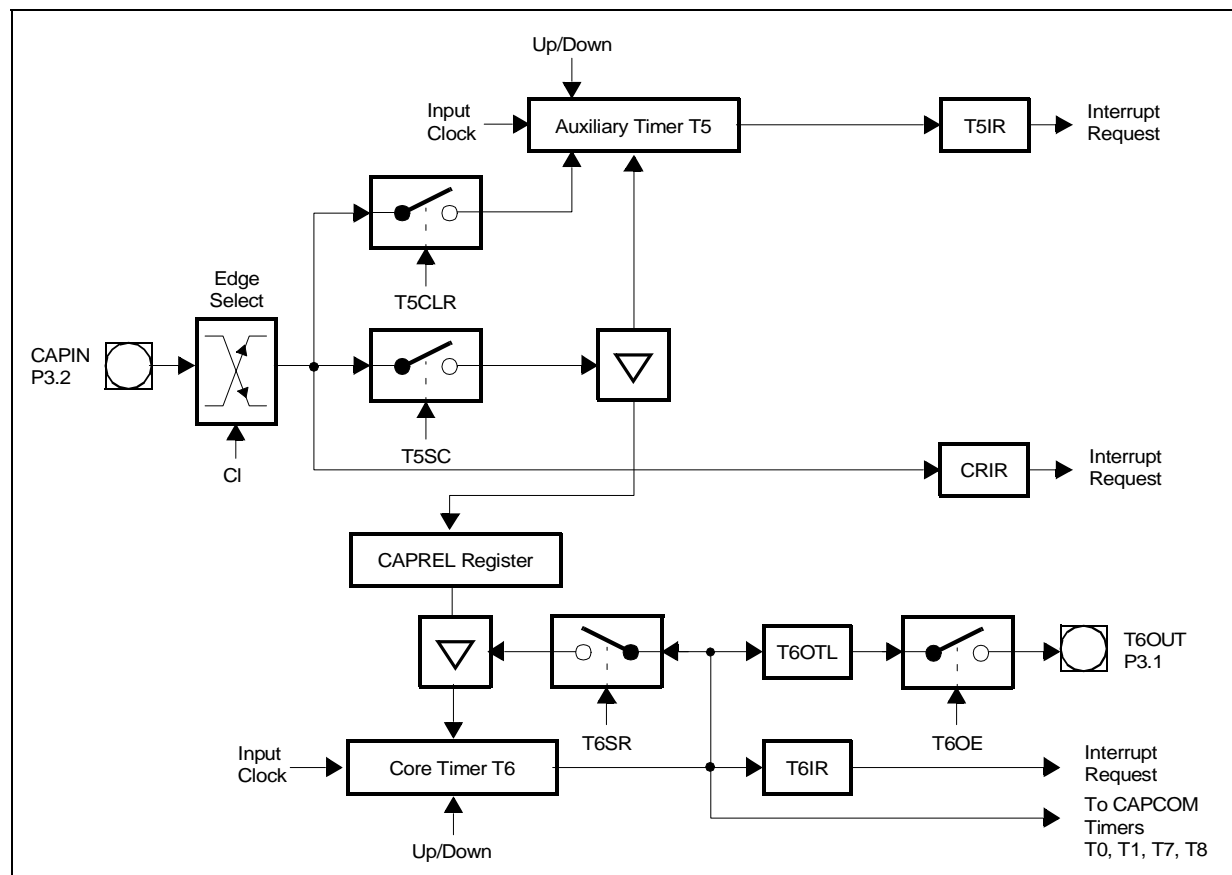
For example, Timer T5 runs in timer mode counting up with a frequency of  $f_{CPU} / 32$ . The external events are applied to pin CAPIN. When an external event occurs, the timer T5 contents are latched into register CAPREL, and timer T5 is cleared (T5CLR='1').

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 86** : GPT2 register CAPREL in capture-and-reload mode



**9.2.2 - 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
-	-	-	-	-	-	-	-	<b>T5IR</b>	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
-	-	-	-	-	-	-	-	<b>T6IR</b>	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
-	-	-	-	-	-	-	-	<b>CRIR</b>	CRIE	ILVL			GLVL		
								RW	RW	RW			RW		

Note Please refer to Interrupt Control Registers for explanation of the control fields.

## 10 - ASYNCHRONOUS/SYNCHRONOUS SERIAL INTERFACE

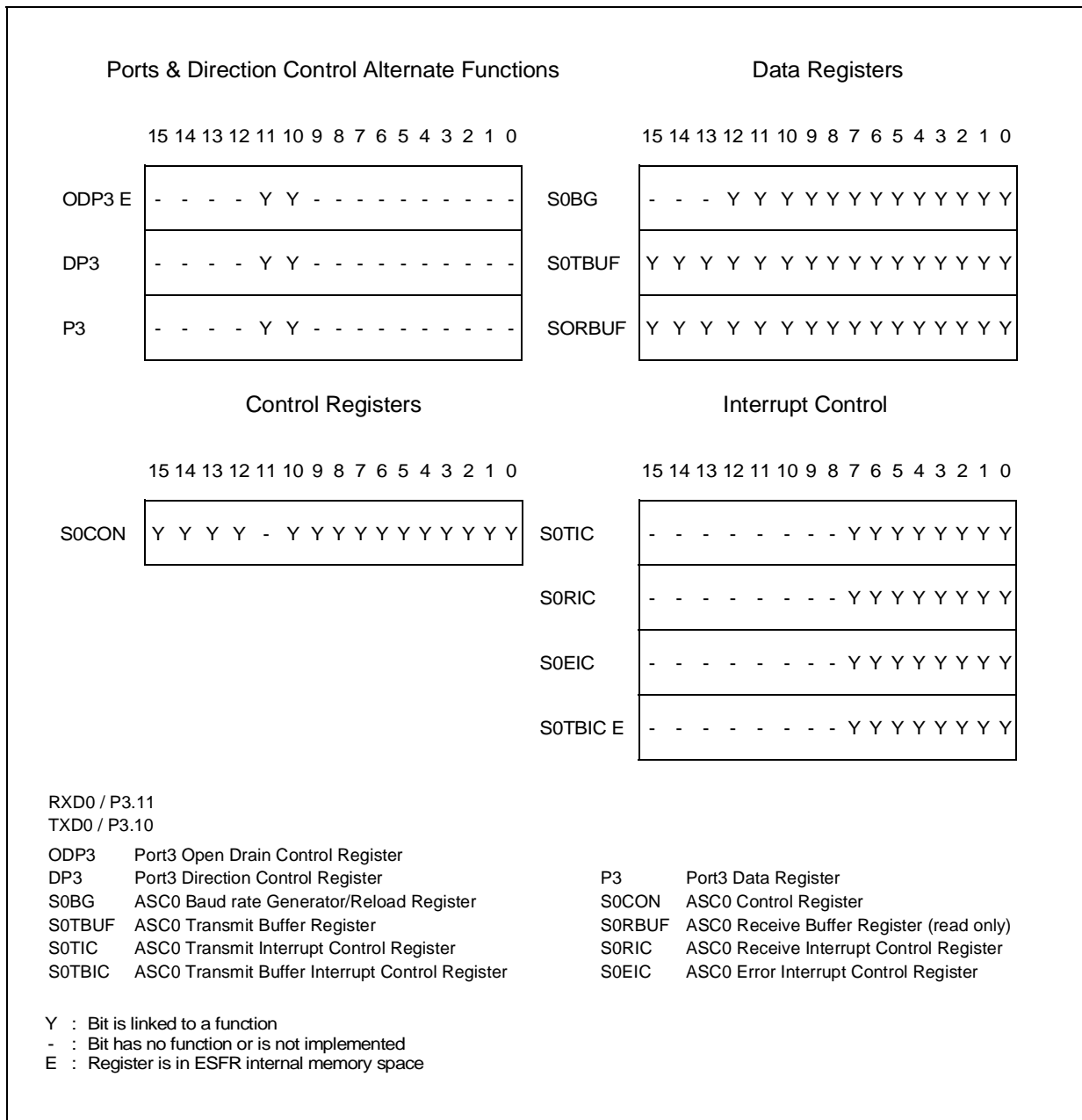
The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between the ST10X167 and other microcontrollers, microprocessors or external peripherals.

In synchronous mode, data are transmitted or received synchronously to a shift clock which is generated by the ST10X167. In asynchronous mode, 8- or 9Bit data transfer, parity generation, and the number of stop Bit 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 87 : 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	<b>ASC0 Mode Control</b> 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	<b>Number of Stop Bit Selection asynchronous operation</b> 0: One stop Bit 1: Two stop Bit
S0REN	<b>Receiver Enable Bit</b> 0: Receiver disabled 1: Receiver enabled (Reset by hardware after reception of Byte in synchronous mode)
S0PEN	<b>Parity Check Enable Bit asynchronous operation</b> 0: Ignore parity 1: Check parity
S0FEN	<b>Framing Check Enable Bit asynchronous operation</b> 0: Ignore framing errors 1: Check framing errors
S0OEN	<b>Overrun Check Enable Bit</b> 0: Ignore overrun errors 1: Check overrun errors
S0PE	<b>Parity Error Flag</b> Set by hardware on a parity error (S0PEN='1'). Must be reset by software.
S0FE	<b>Framing Error Flag</b> Set by hardware on a framing error (S0FEN='1'). Must be reset by software.
S0OE	<b>Overrun Error Flag</b> Set by hardware on an overrun error (S0OEN='1'). Must be reset by software.
S0ODD	<b>Parity Selection Bit</b> 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	<b>Baud rate Selection Bit</b> 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3rd
S0LB	<b>Loopback Mode Enable Bit</b> 0: Standard transmit/receive mode 1: Loopback mode enabled
S0R	<b>Baud rate Generator Run Bit</b> 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.

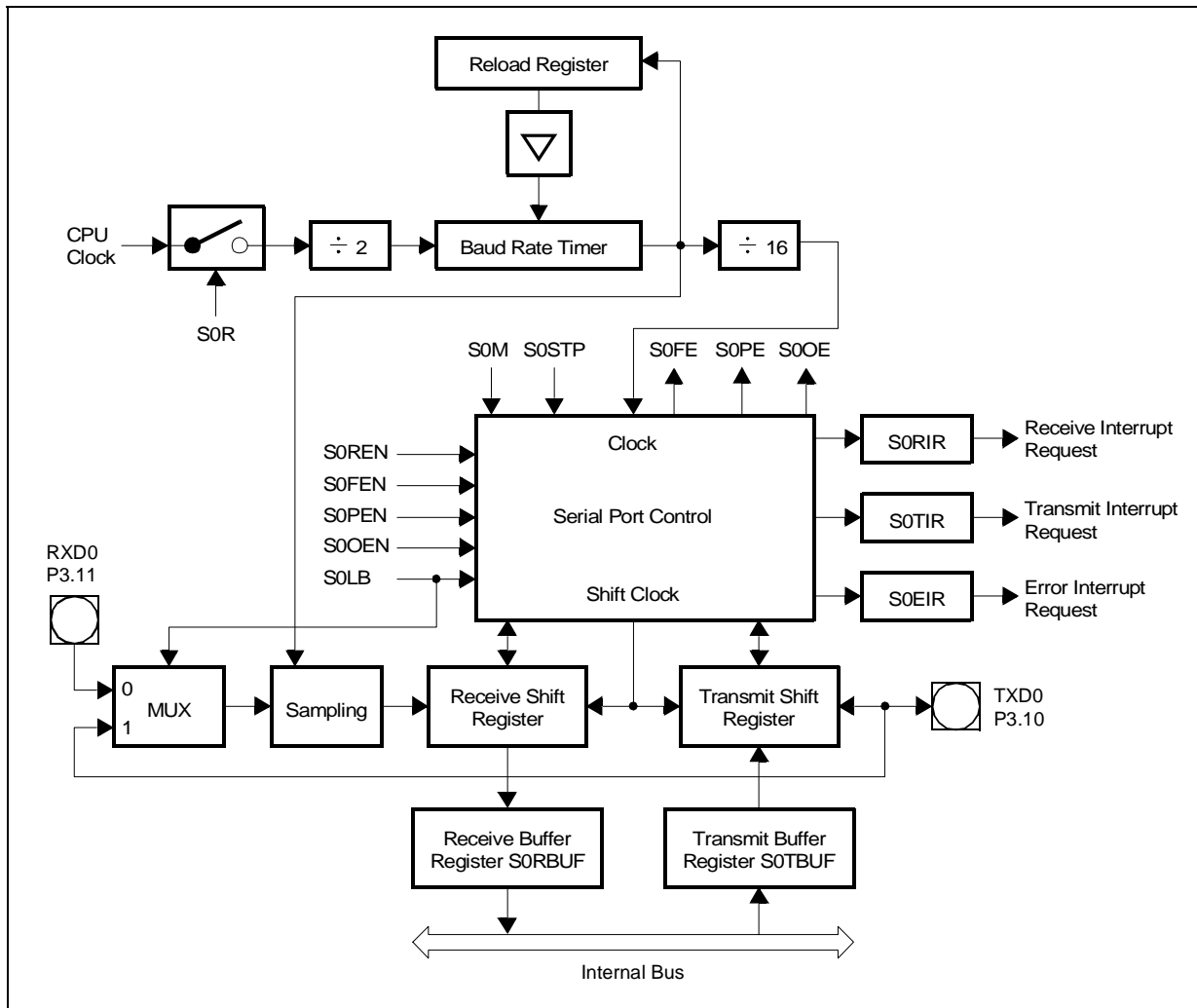
This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output functions of the Port3 pins are not necessary.

**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.

### **10.1 - 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 88 : Asynchronous mode of serial channel ASC0

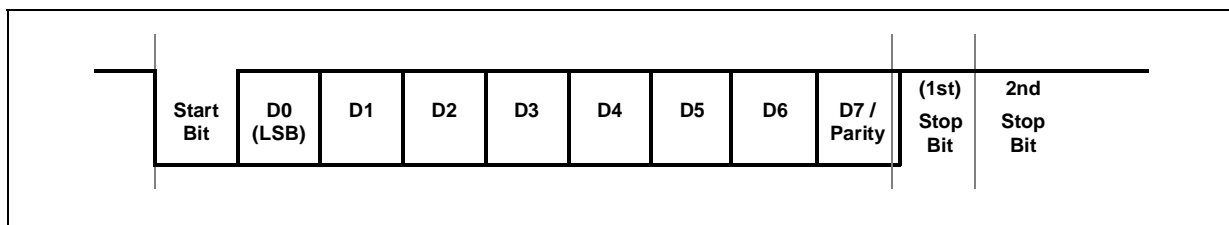


**Asynchronous Data Frames**

**8 Bit data frames** either consist of 8 data Bit D7...D0 (SOM='001b'), or of 7 data Bit D6...D0 plus an automatically generated parity Bit (SOM='011b'). Parity may be odd or even, depending on Bit SOODD in register SOCON. 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 SORPEN (always OFF in 8 Bit data mode). The parity error flag SOPE 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 SORBUF.7.

Figure 89 : Asynchronous 8 Bit data frames



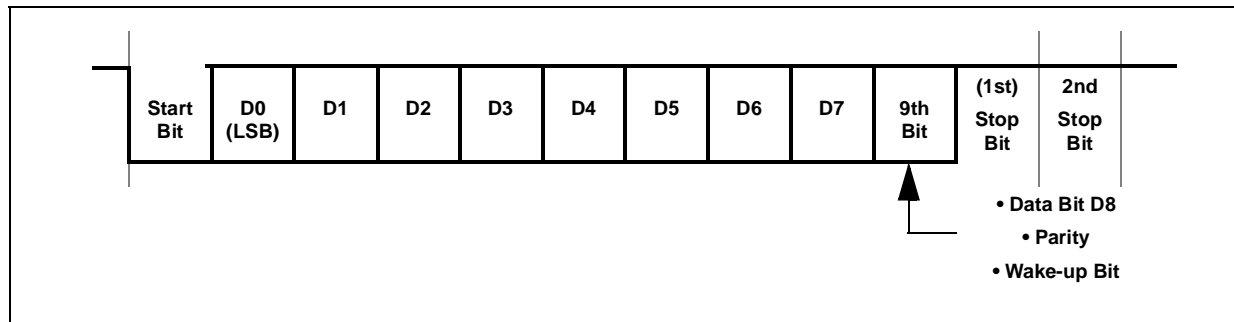
**9 Bit data frames** either consist of 9 data Bit D8...D0 (SOM='100b'), of 8 data Bit D7...D0 plus an automatically generated parity Bit (SOM='111b') or of 8 data Bit D7...D0 plus wake-up Bit (SOM='101b'). Parity may be odd or even, depending on Bit S0ODD in register S0CON. An even parity Bit will be 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 90).

**Figure 90** : Asynchronous 9 Bit data frames



**Asynchronous transmission** begins at the next overflow of the divide-by-16 counter (see Figure 90), 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.



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.

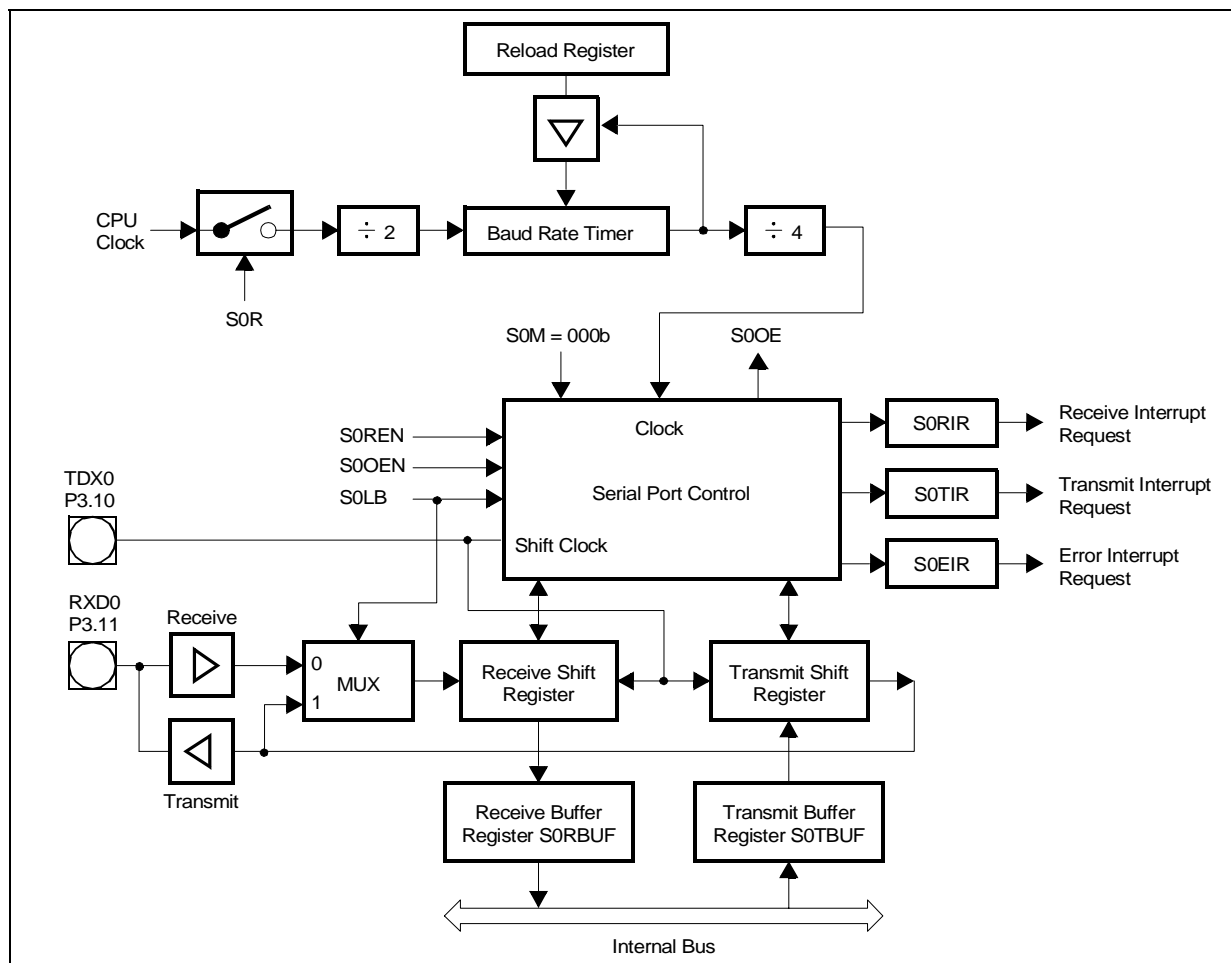
**Note** In wake-up mode received frames are only transferred to the receive buffer register, if the 9th Bit (the wake-up Bit) is '1'. If this Bit is '0', no receive interrupt request will be activated and no data will be transferred.

## 10.2 - 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 Bit 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 Bit are transmitted or received.

**Figure 91** : 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 Bit 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.

### 10.3 - 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).

### 10.4 - 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 SOBGR is written to. However, if SOR=‘0’ at the time the write operation to SOBGR is performed, the timer will not be reloaded until the first instruction cycle after SOR=‘1’.

**Asynchronous Mode Baud rates**

For asynchronous operation, the Baud rate generator provides a clock with 16 times the rate of the established Baud rate. Every received Bit is sampled at the 7th, 8th and 9th cycle of this clock. The Baud rate for asynchronous operation of serial channel ASC0 and the required reload value for a given Baud rate can be determined by the following formulas:

$$B_{Async} = \frac{f_{CPU}}{16 \times [2 + (SOBRS)] \times [(SOBRL) + 1]}$$

$$SOBRL = \left( \frac{f_{CPU}}{16 \times [2 + (SOBRS)] \times B_{Async}} \right) - 1$$

(SOBRL) represents the content of the reload register, taken as unsigned 13 Bit integer, (SOBRS) represents the value of Bit SOBRS (‘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:

$$B_{Sync} = \frac{f_{CPU}}{4 \times [2 + (SOBRS)] \times [(SOBRL) + 1]}$$

$$SOBRL = \left( \frac{f_{CPU}}{4 \times [2 + (SOBRS)] \times B_{Sync}} \right) - 1$$

(SOBRL) represents the content of the reload register, taken as unsigned 13 Bit integers, (SOBRS) represents the value of Bit SOBRS (‘0’ or ‘1’), taken as integer.

Using the above equation, the maximum Baud rate can be calculated for any given clock speed.

**10.5 - 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 5.1.3 - Interrupt Control Registers for an explanation of the control fields.

the previously loaded data has been transmitted, except for the last Bit of an asynchronous frame.

**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:

- S0TBIR is activated when data is moved from S0TBUF to the transmit shift register.
- S0TIR is activated before the last Bit of an asynchronous frame is transmitted, or after the last Bit of a synchronous frame has been transmitted.
- S0RIR is activated when the received frame is moved to S0RBUF.

**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.

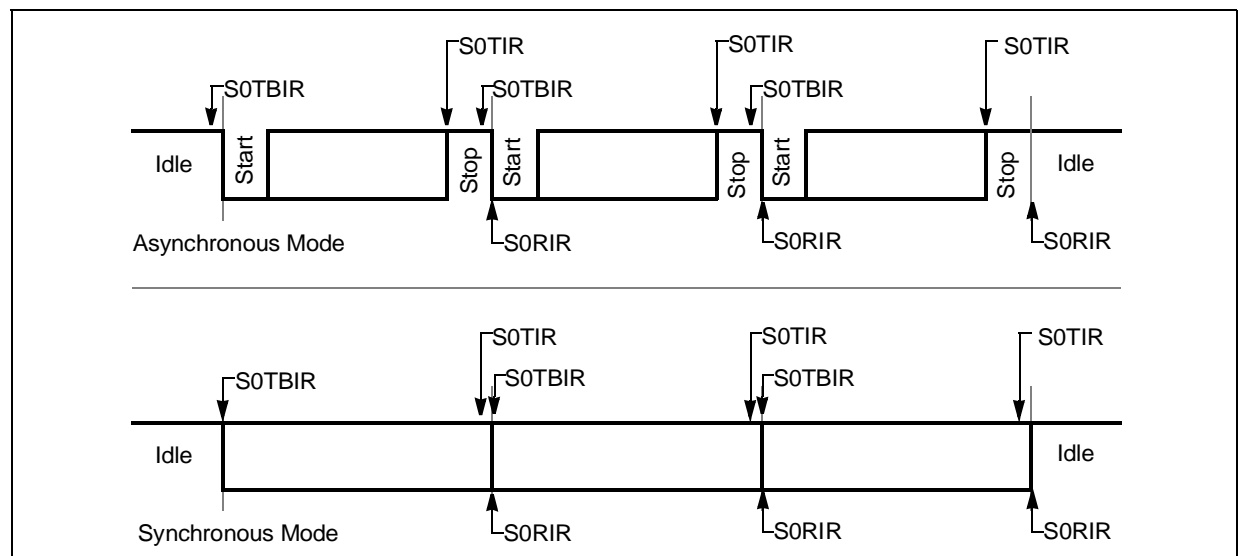
Using the transmit buffer interrupt (S0TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

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.

As shown in the Figure 92, 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.

**For single transfers** is sufficient to use the transmitter interrupt (S0TIR), which indicates that

**Figure 92 : ASC0 interrupt generation**



## 11 - HIGH-SPEED SYNCHRONOUS SERIAL INTERFACE

The High-Speed Synchronous Serial Interface SSC provides flexible high-speed serial communication between the ST10X167 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 93 : SFRs and port pins associated with the SSC

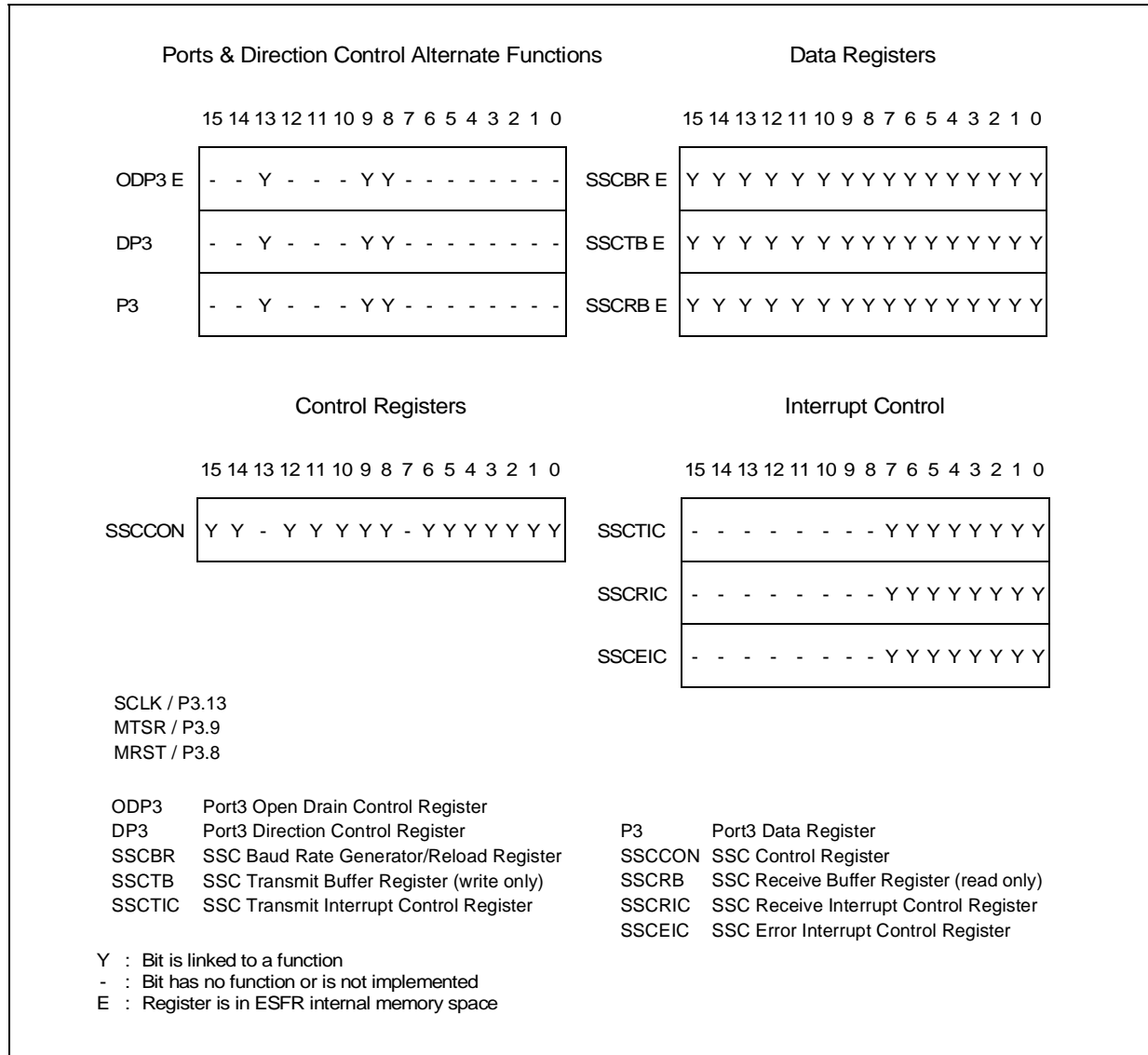
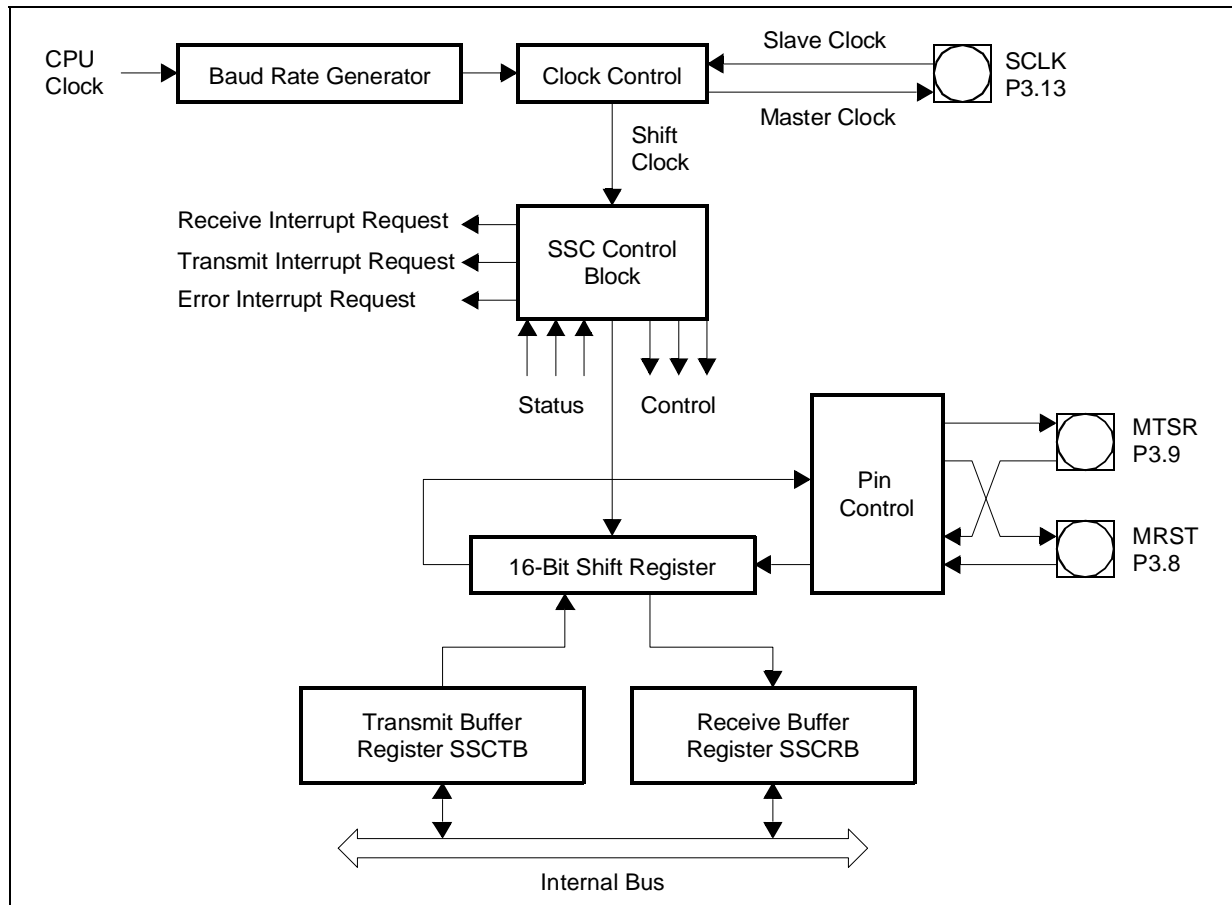


Figure 94 : Synchronous serial channel SSC block diagram



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.

## ST10X167

### SSCCON (FFB2h / D9h)

SFR

ResetValue: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	<b>SSC Data Width Selection</b> 0: Reserved. Do not use this combination. 1...15: Transfer Data Width is 2...16 Bit [(SSCBM)+1]
SSCHB	<b>SSC Heading Control Bit</b> 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
SSCPH	<b>SSC Clock Phase Control Bit</b> 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	<b>SSC Clock Polarity Control Bit</b> 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	<b>SSC Transmit Error Enable Bit</b> 0: Ignore transmit errors 1: Check transmit errors
SSCEN	<b>SSC Receive Error Enable Bit</b> 0: Ignore receive errors 1: Check receive errors
SSCPEN	<b>SSC Phase Error Enable Bit</b> 0: Ignore phase errors 1: Check phase errors
SSCBEN	<b>SSC Baudrate Error Enable Bit</b> 0: Ignore baudrate errors 1: Check baudrate errors
SSCAREN	<b>SSC Automatic Reset Enable Bit</b> 0: No additional action upon a baudrate error 1: The SSC is automatically reset upon a baudrate error
SSCMS	<b>SSC Master Select Bit</b> 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	<b>SSC Enable Bit = '0'</b> 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	<b>SSC Bit Count Field</b> Shift counter is updated with every shifted Bit. <b>Do not write to</b>
SSCTE	<b>SSC Transmit Error Flag</b> 1: Transfer starts with the slave's transmit buffer not being updated
SSCRE	<b>SSC Receive Error Flag</b> 1: Reception completed before the receive buffer was read
SSCPE	<b>SSC Phase Error Flag</b> 1: Received data changes around sampling clock edge
SSCBE	<b>SSC Baud rate Error Flag</b> 1: More than factor 2 or 0.5 between Slave's actual and expected Baud rate
SSCBSY	<b>SSC Busy Flag:</b> Set while a transfer is in progress. <b>Do not write to</b>
SSCMS	<b>SSC Master Select Bit</b> 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	<b>SSC Enable Bit = '1'</b> 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 94). 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 Bit to 16 Bit.
- 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 11.3 - 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).

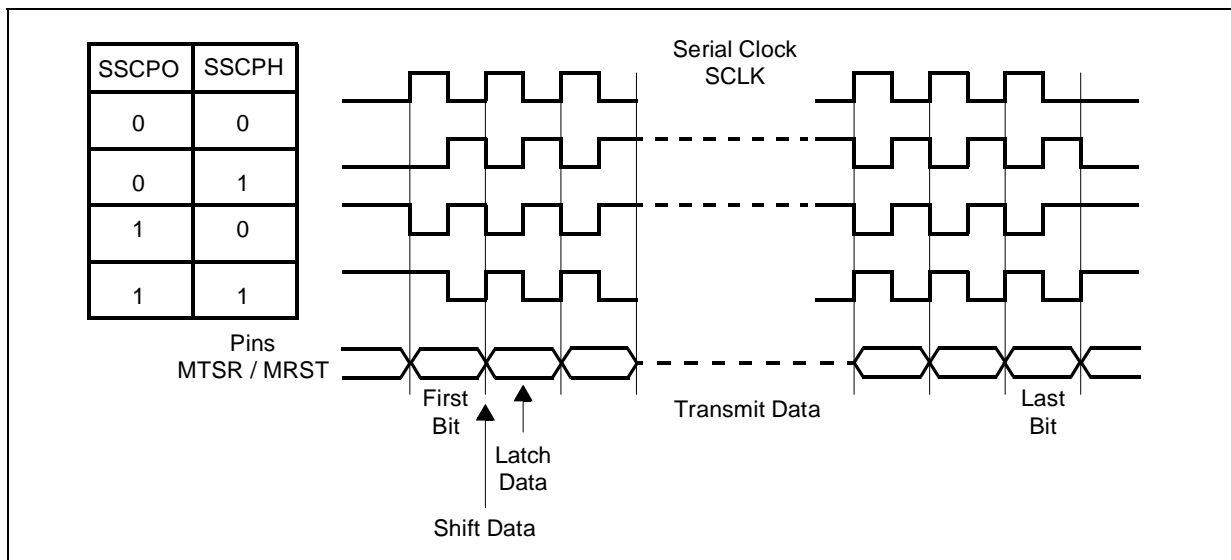
This allows the adaptation of the SSC to a wide range of applications, where serial data transfer is required.

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 Bit are rearranged for transfer by the internal shift register logic. The unselected Bit of SSCTB are ignored, the unselected Bit 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 95 is a summary.

Figure 95 : Serial clock phase and polarity options



11.1 - Full-Duplex Operation

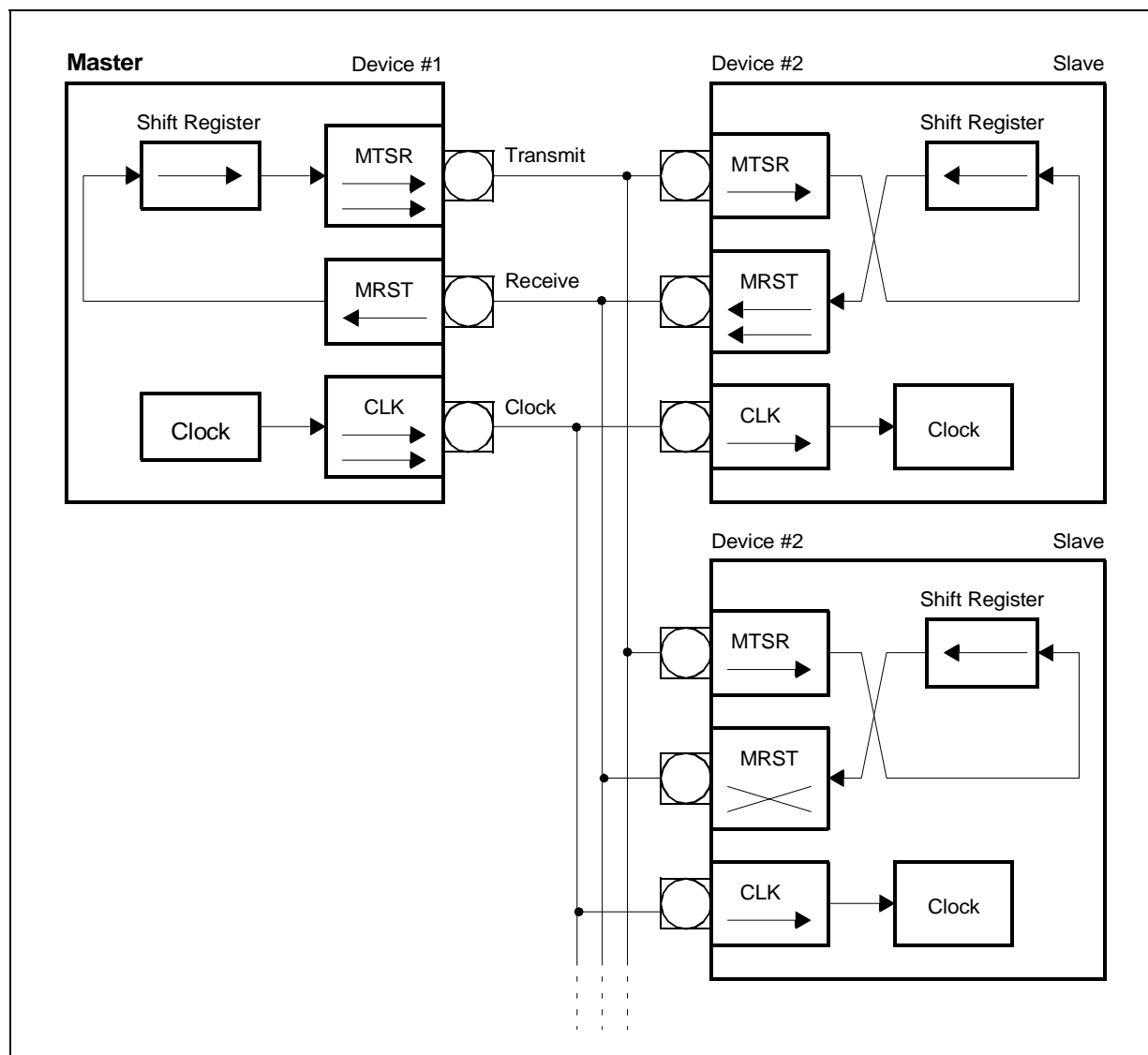
The different devices are connected through three lines. The definition of these lines is always determined by the master: The line connected to the master’s data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. 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 96 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 11.2.1 - Port Control).

Figure 96 : 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 their 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 an 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).

## 11.2 - 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.

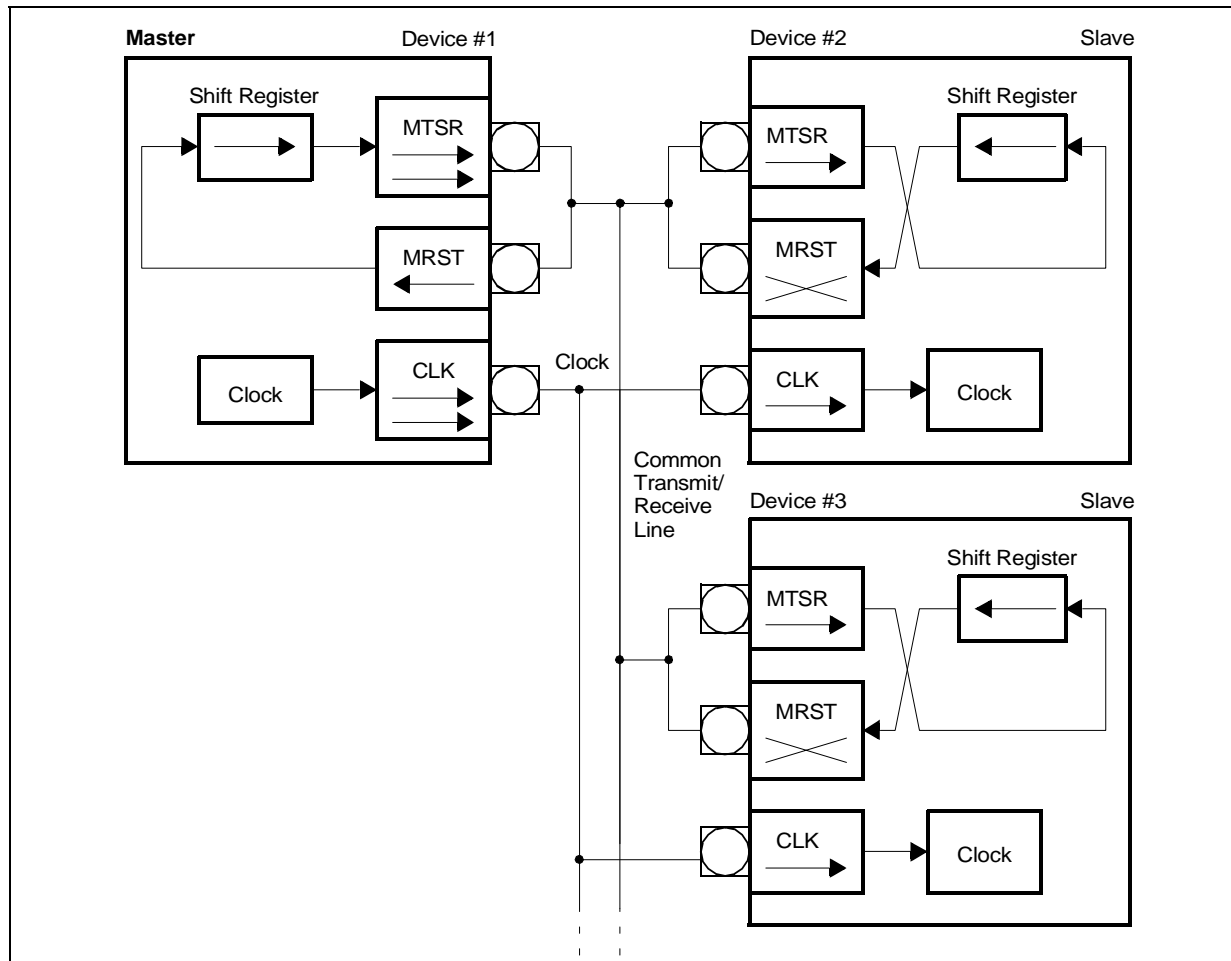
The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

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 97 : 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 Byte transfers would look the same as one Word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data Bit 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.

### 11.2.1 - 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 Bit to '1', so they are already in the correct state when switching between master and slave mode.

### 11.3 - 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.

### 11.4 - 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 98).

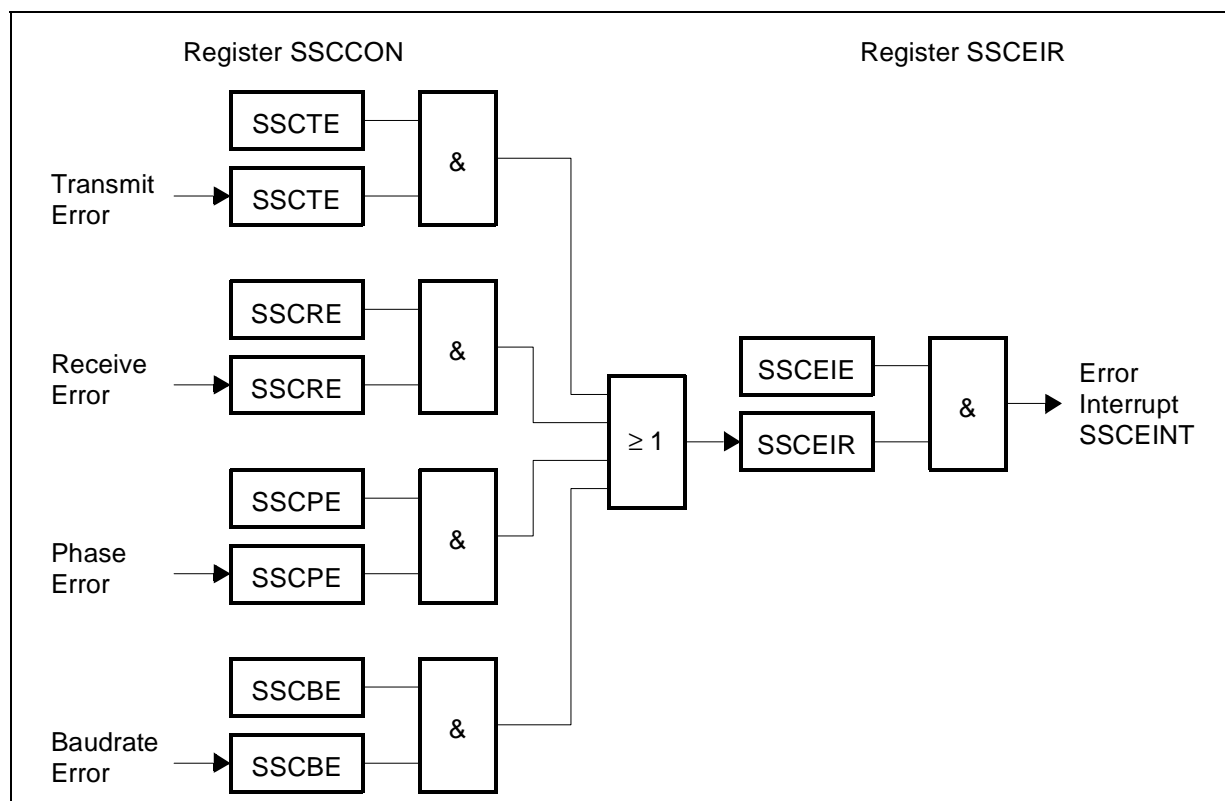
### 11.5 - SSC Interrupt Control

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 98** : SSC error interrupt control



## ST10X167

---

### SSCTIC (FF72h / B9h)

SFR

ResetValue:--00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	<b>SSC TIR</b>	SSC TIE	ILVL			GLVL		
								RW	RW	RW			RW		

### SSCRIC (FF74h / BAh)

SFR

ResetValue:--00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	<b>SSC RIR</b>	SSC RIE	ILVL			GLVL		
								RW	RW	RW			RW		

### SSCEIC (FF76h / BBh)

SFR

ResetValue:--00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	<b>SSC EIR</b>	SSC EIE	ILVL			GLVL		
								RW	RW	RW			RW		

Note Please refer to Section 5.1.3 - Interrupt Control Registers for an explanation of the control fields.



## 12 - 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.

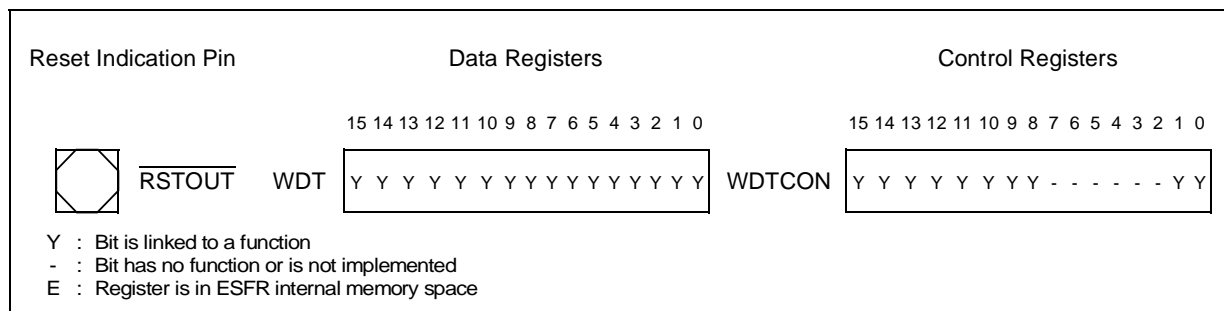
The watchdog timer provides two registers:

- Read-only timer register that contains the current count.
- Control register for initialization.

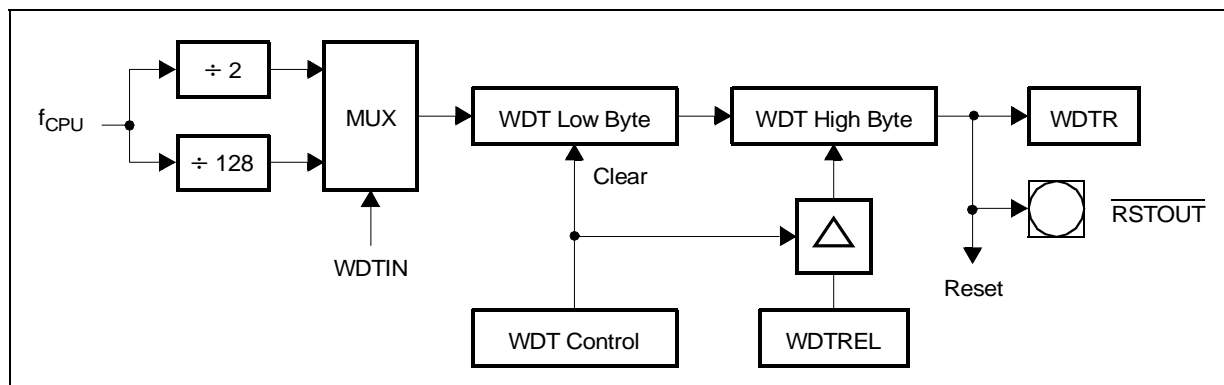
The watchdog timer is a 16 Bit up counter which can be clocked with the CPU clock ( $f_{\text{CPU}}$ ) either divided by 2 or divided by 128. This 16 Bit timer is realized as two concatenated 8 Bit timers (see Figure 100).

The upper 8 Bit 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 Bit are reset on each service access.

**Figure 99** : SFRs and port pins associated with the watchdog timer



**Figure 100** : Watchdog timer block diagram



### 12.1 - Operation of the Watchdog Timer

The current count value of the Watchdog Timer is contained in the Watchdog Timer Register WDT, which is a Bitaddressable read-only register. The operation of the Watchdog Timer is controlled by its Bitaddressable Watchdog Timer Control Register WDTCON. This register specifies the reload value for the high Byte of the timer, selects the input clock prescaling factor and provides a flag that indicates a watchdog timer overflow.

<b>WDTCN (FFAEh / D7h)</b>											<b>SFR</b>			ResetValue:000Xh				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
WDTREL											-	-	-	-	-	-	<b>WDTR</b>	<b>WDTIN</b>
RW																	RW	RW

Bit	Function
WDTIN	<b>Watchdog Timer Input Frequency Selection</b> '0': Input frequency is $f_{CPU} / 2$ '1': Input frequency is $f_{CPU} / 128$
WDTR	<b>Watchdog Timer Reset Indication Flag</b> Set by the watchdog timer on an overflow. Cleared by a hardware reset or by the SRVWDT instruction.
WDTREL	<b>Watchdog Timer Reload Value</b> (for the high Byte)

**Note** The reset value will be 0002h, if the reset was triggered by the watchdog timer (overflow). It will be 0000h otherwise.

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 RSTOUT low. It differs from a software or external hardware reset in that Bit WDTR (Watchdog Timer Reset Indication Flag) of register WDTCN 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 READY or samples READY active (low) after the programmed waitstates. Otherwise the external bus cycle will be aborted.

After a hardware reset that activates the Bootstrap Loader the watchdog timer will be disabled.

To prevent the watchdog timer from overflowing, it must be serviced periodically by the user software. The watchdog timer is serviced with the instruction SRVWDT, which is a protected 32 Bit instruction. Servicing the watchdog timer clears the low Byte and reloads the high Byte of the watchdog time register WDT with the preset value in Bit field WDTREL, which is the high Byte of register WDTCN. Servicing the watchdog timer will also reset Bit WDTR.

After being serviced the watchdog timer continues counting up from the value  $[(WDTREL) \times 28]$ . Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer (eg. by fetching and executing a Bit pattern from a wrong location) is minimized. When instruction SRVWDT does not match the format for protected instructions, the Protection Fault Trap will be entered, rather than the instruction be executed.

The time period for an overflow of the watchdog timer is programmable in two ways:

- **The input frequency** to the watchdog timer can be selected via Bit WDTIN in register WDTCN 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 WDTCN.

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 WDTCN each time before the watchdog timer is serviced.



### 13 - BOOTSTRAP LOADER

The built-in bootstrap loader of the ST10X167 provides a mechanism to load the startup program through the serial interface after reset. In this case, no external or internal ROM 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. ROM 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 101).

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 (firmware update) or end-of-line programming or testing.

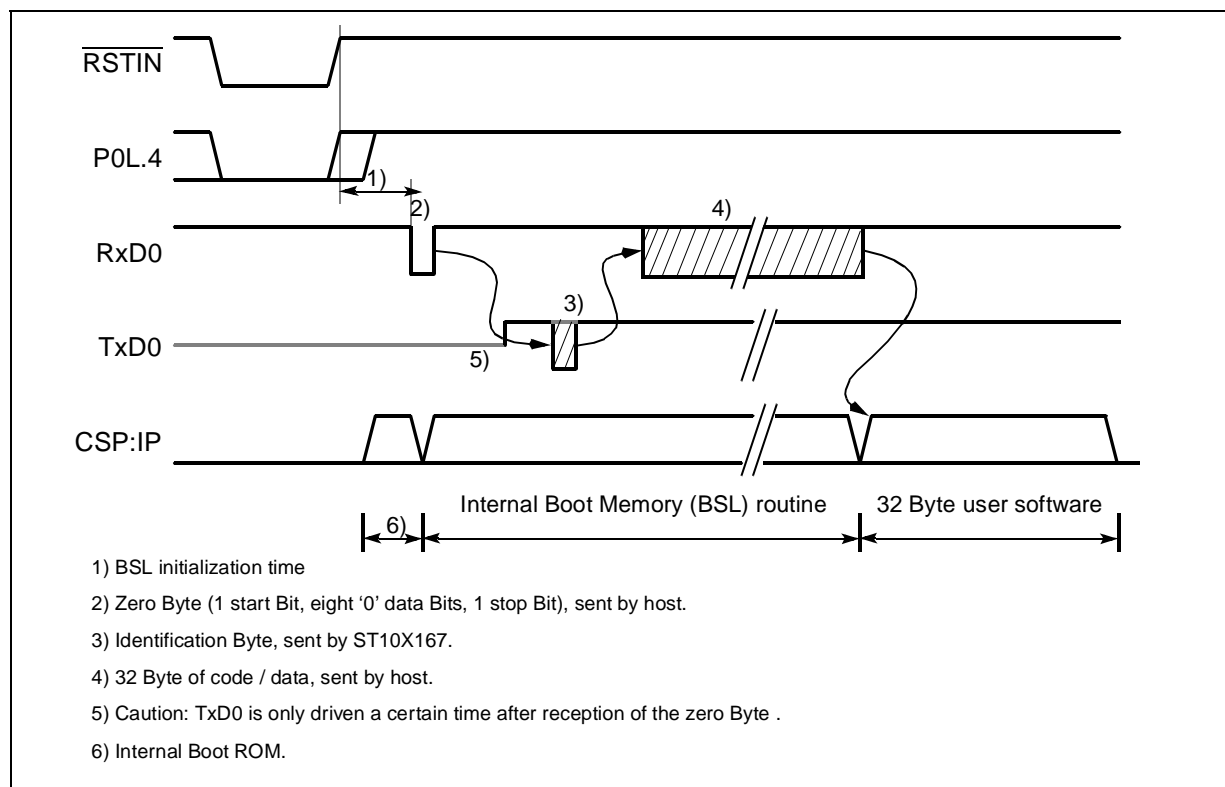
#### Entering the bootstrap loader

The ST10X167 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 ST10X167 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 101** : Bootstrap loader sequence



When the ST10X167 has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are **marked**):

Watchdog Timer:	<b>Disabled</b>	Register SYSCON:	0E00h
Context Pointer CP:	FA00h	Register STKUN:	FA40h
Stack Pointer SP:	FA40h	Register STKOV:	FA0Ch 0<->C
Register S0CON:	<b>8011h</b>	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 ST10X167 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 POL.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 102).

After sending the identification Byte the ASC0 receiver is enabled and is ready to receive the initial 32 Byte 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 ST10X167's memory areas after reset in Bootstrap-Loader mode differs from the standard case. Pin EA is not evaluated when BSL mode is selected, and accesses to the internal Flash area are partly redirected, while the ST10X167 is in BSL mode (see Figure 103). All code fetches are made from the special Boot-ROM, while data accesses will return undefined values on ROMless devices.

The code in the Boot-ROM is not an invariant feature of the ST10X167. User software should not try to execute code from the internal ROM area while the BSL mode is still active, as these fetches will be redirected to the Boot-ROM. The Boot-ROM will also "move" to segment 1, when the internal ROM area is mapped to segment 1 (see Figure 103).

**Figure 102** : Hardware provisions to activate the BSL

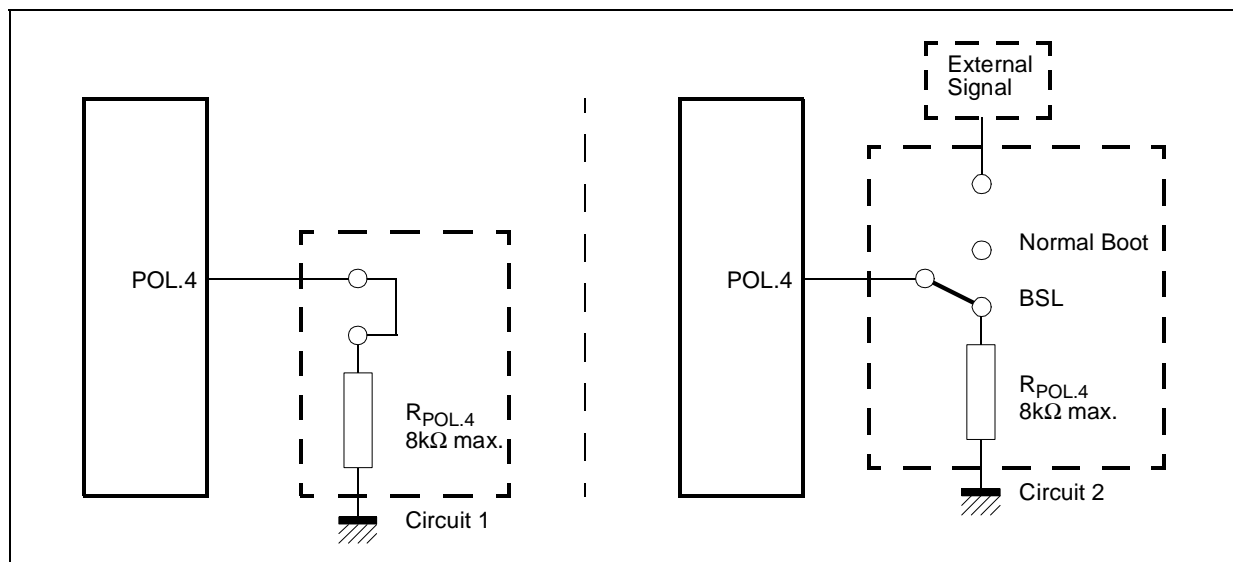


Figure 103 : Memory configuration after reset

BSL mode active	Yes (POL.4='0')	Yes (POL.4='0')	No (POL.4='1')
$\overline{EA}$ pin	high	low	according to application
Code fetch from internal ROM area	Boot-ROM access	Boot-ROM access	User ROM access
Data fetch from internal ROM area	User ROM access	User ROM access	User ROM access

### Loading the Startup Code

After sending the identification Byte the BSL enters a loop to receive 32 Byte via ASC0. These Byte 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 ST10X167 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 ST10X167 will still run in BSL mode, that means with the watchdog timer disabled and limited access to the internal ROM area.

All code fetches from the internal ROM 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 ST10X167, 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 ST10X167 exits BSL mode upon a software reset (ignores the level on POL.4) or a hardware reset (POL.4 must be high). After a reset the ST10X167 will start executing from location 00'0000h of the internal ROM 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 ST10X167 with a wide range of Baud rates. However, the upper and lower limits have to be kept, in order to insure proper data transfer.

$$B_{ST10X167} = \frac{f_{CPU}}{32 \times (S0BRL + 1)}$$

The ST10X167 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:

$$S0BRL = \frac{T6 - 36}{72}, \quad T6 = \frac{9}{4} \times \frac{f_{CPU}}{B_{Host}}$$

For a correct data transfer from the host to the ST10X167 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 ST10X167 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.

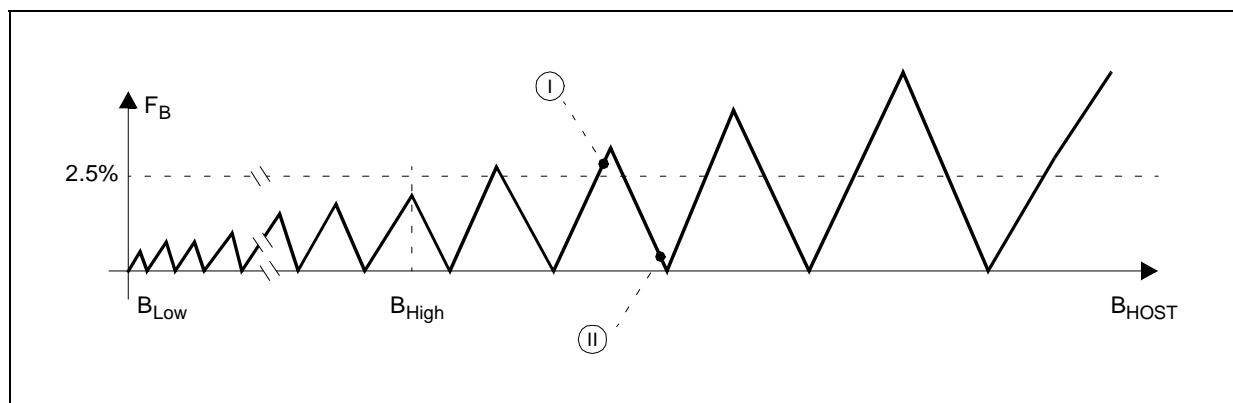
This Baud rate deviation is a nonlinear function depending on the CPU clock and the Baud rate of the host. The maxima of the function ( $F_B$ ) increase with the host Baud rate due to the smaller Baud rate pre-scaler factors and the implied higher quantization error (see Figure 104).

**The minimum Baud rate** ( $B_{Low}$  in the Figure 104) 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 104) 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 104) may violate the deviation limit, while an even higher Baud rate (marked 'II' in Figure 104) stays very well below it. This depends on the host interface.

**Figure 104 :** Baud rate deviation between host and ST10X167



#### 14 - THE CAPTURE / COMPARE UNITS

The ST10X167 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 Bit.





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 Chapter Section 14.1 - CAPCOM Timers and is specified in the device datasheet.

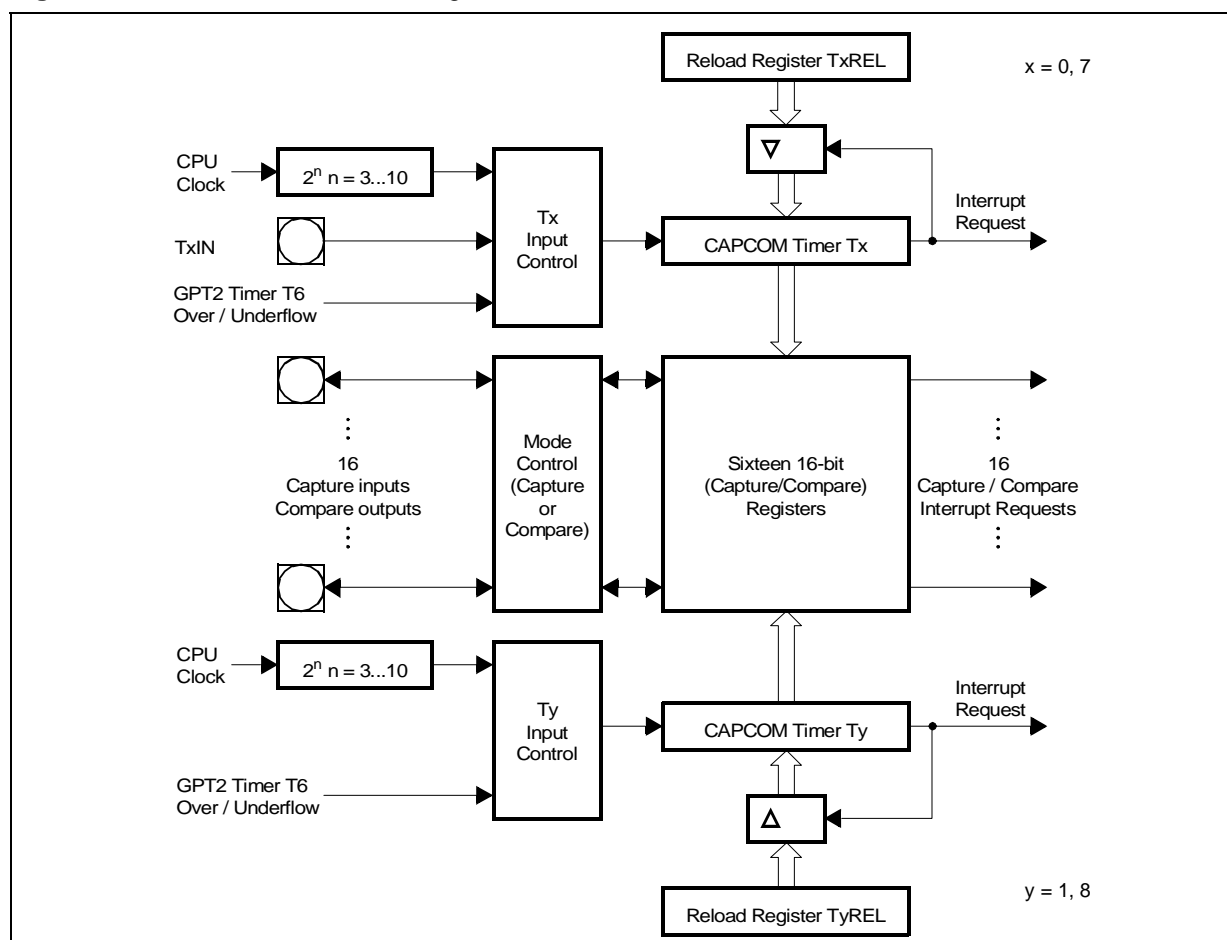
Each CAPCOM unit consists of two 16 Bit timers (T0 / T1 in CAPCOM1, T7 / T8 in CAPCOM2), each with its own reload register (TxREL), and a bank of sixteen dual purpose 16 Bit capture / compare registers (CC0 through CC15 in CAPCOM1, CC16 through CC31 in CAPCOM2).

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 106 shows the basic structure of the two CAPCOM units.

Figure 106 : CAPCOM unit block diagram



Note The CAPCOM2 unit provides 16 capture inputs, but only 12 compare outputs.



14.1 - 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 107 : Block diagram of CAPCOM timers T0 and T7

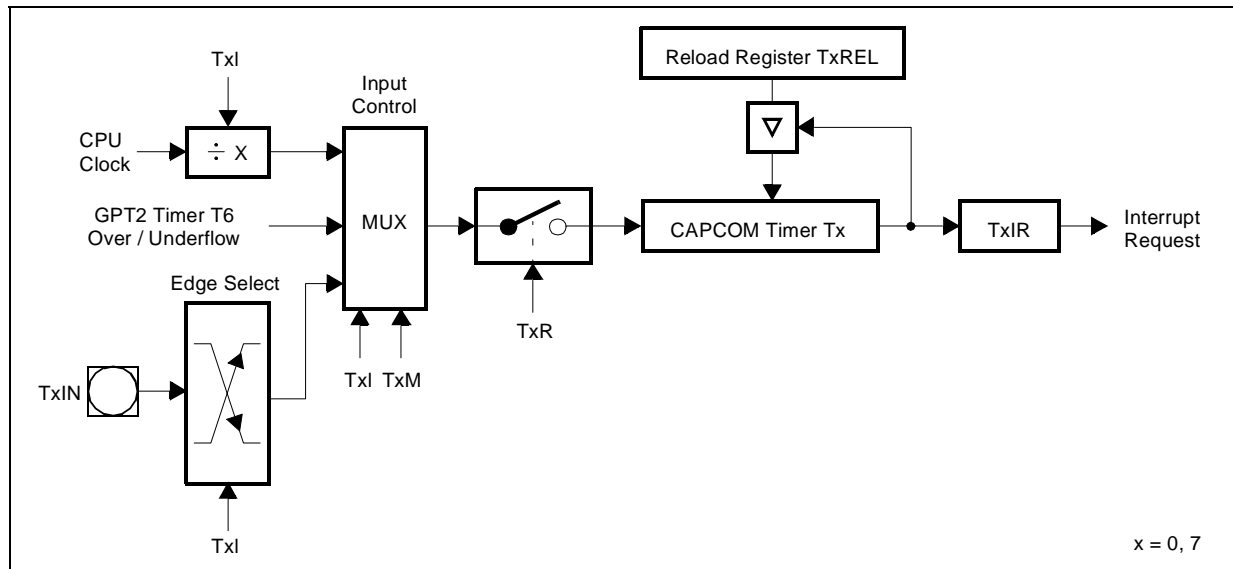
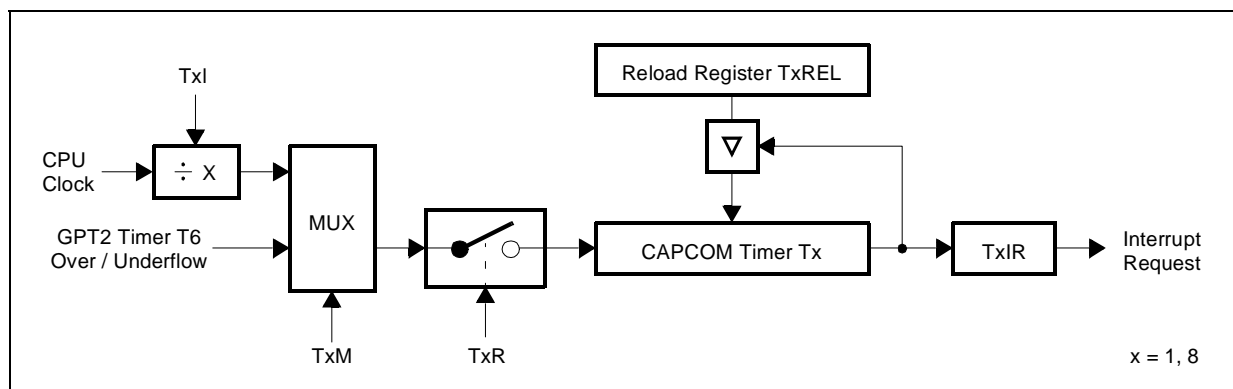


Figure 108 : Block diagram of CAPCOM timers T1 and T8



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		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		RW					

Bit	Function
TxI	<b>Timer / Counter x Input Selection</b> 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 <sup>1)</sup> X10 Negative (falling) edge on pin TxIN <sup>1)</sup> X11 Any edge (rising and falling) on pin TxIN <sup>1)</sup>
TxM	<b>Timer / Counter x Mode Selection</b> '0': Timer Mode (Input derived from internal clock) '1': Counter Mode (Input from External Input or T6)
TxR	<b>Timer / Counter x Run Control</b> '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 Bitaddressable 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 guarantee 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.

**14.2 - 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 Bitaddressable 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.

**14.3 - 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 Bitaddressable.

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 8 Bit addressable 16Bit mode control registers named CCM0...CCM7 which are all organized identically (see description below).

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	<b>Mode Selection for Capture / Compare Register CCx</b> The available capture / compare modes are listed in the table below.
ACCx	<b>Allocation Bit for Capture / Compare Register CCx</b> '0': CCx allocated to Timer T0 (CAPCOM1) / Timer T7 (CAPCOM2) '1': CCx allocated to Timer T1 (CAPCOM1) / Timer T8 (CAPCOM2)

### 14.3.1 - 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, Bit 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 ST10X167's A / D converter if enabled.

### 14.4 - Capture Mode

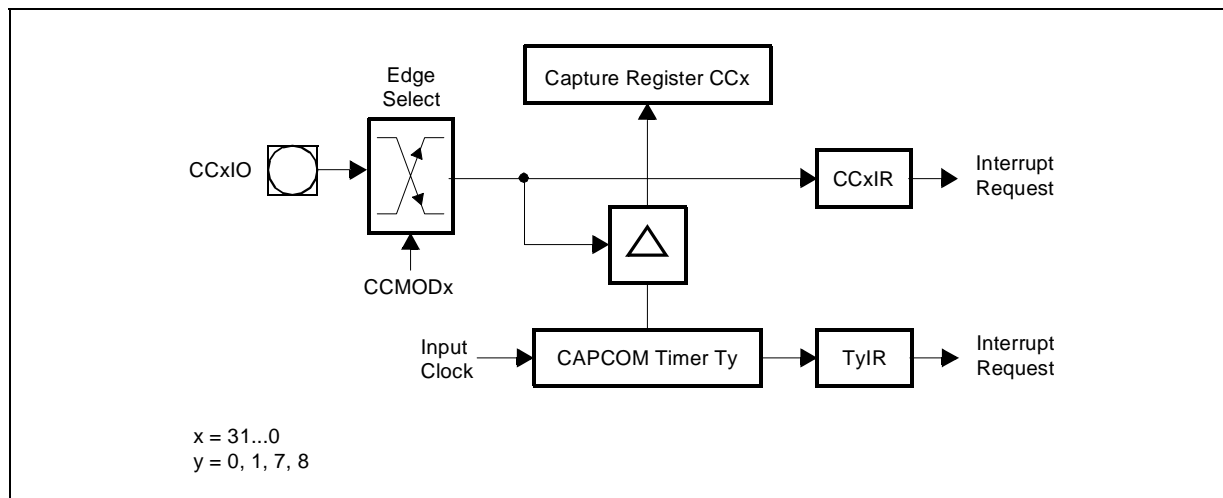
In response to an external event the content of the associated timer (T0 / T1 or T7 / T8, depending on the used CAPCOM unit and the state of the allocation control Bit ACCx) is latched into the respective capture register CCx. The external event causing a capture can be programmed to be either a positive, 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 109).

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 109 : Capture mode block diagram



**14.5 - 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 30 : 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.



### 14.5.1 - 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.

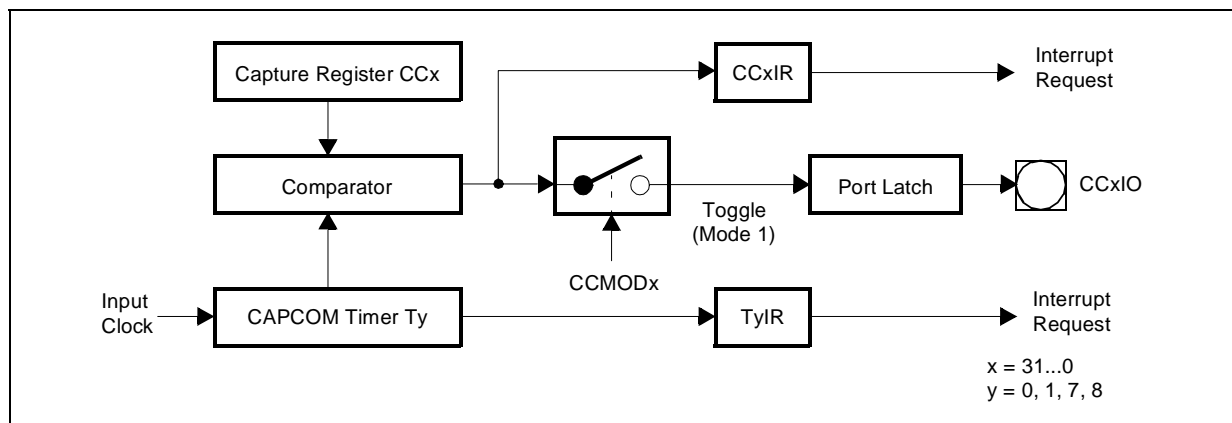
Several of these compare events are possible within a single timer period, when the compare

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 110** : 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 111).

**14.5.2 - 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 111).

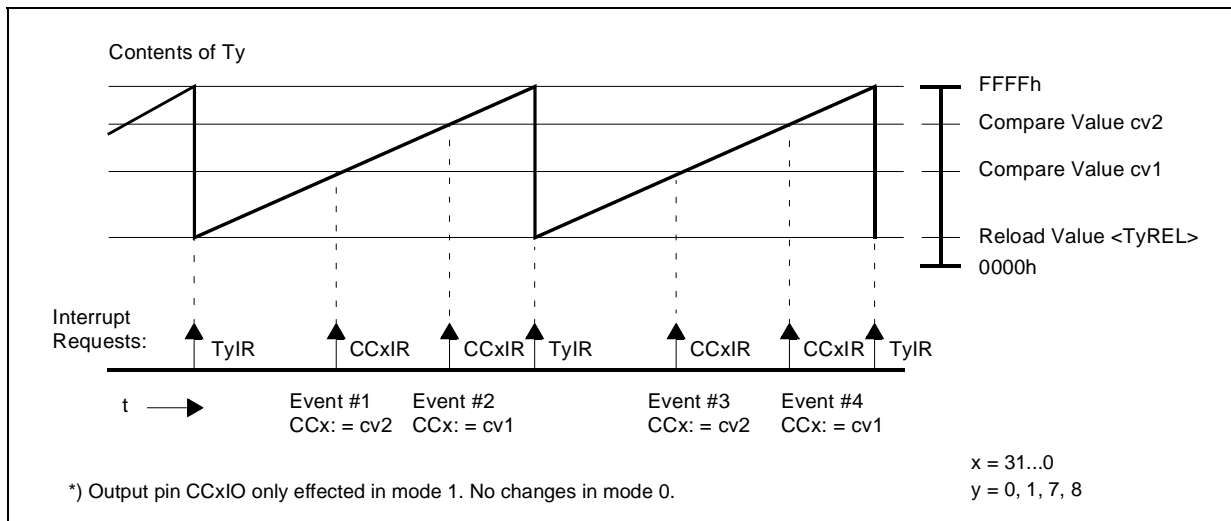
**Note** If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.

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 111 : Timing example for compare modes 0 and 1**



**14.5.3 - Compare Mode 2**

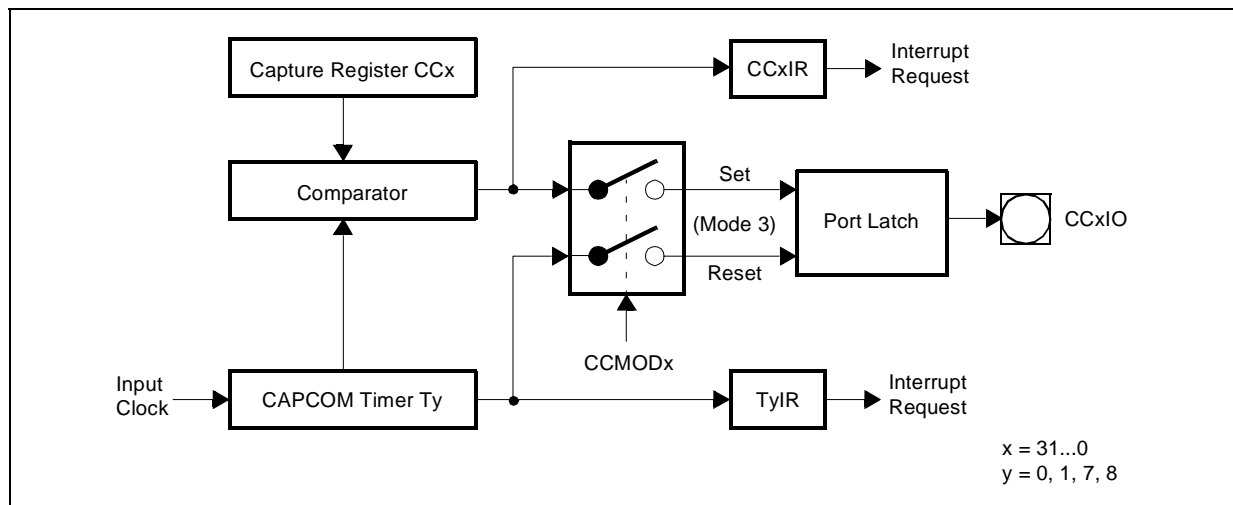
Compare mode 2 is an interrupt-only mode similar to compare mode 0, but only one interrupt request per timer period will be generated. Compare mode 2 is selected for register CCx by setting Bit field CCMODx of the corresponding mode control register to '110b'.

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.

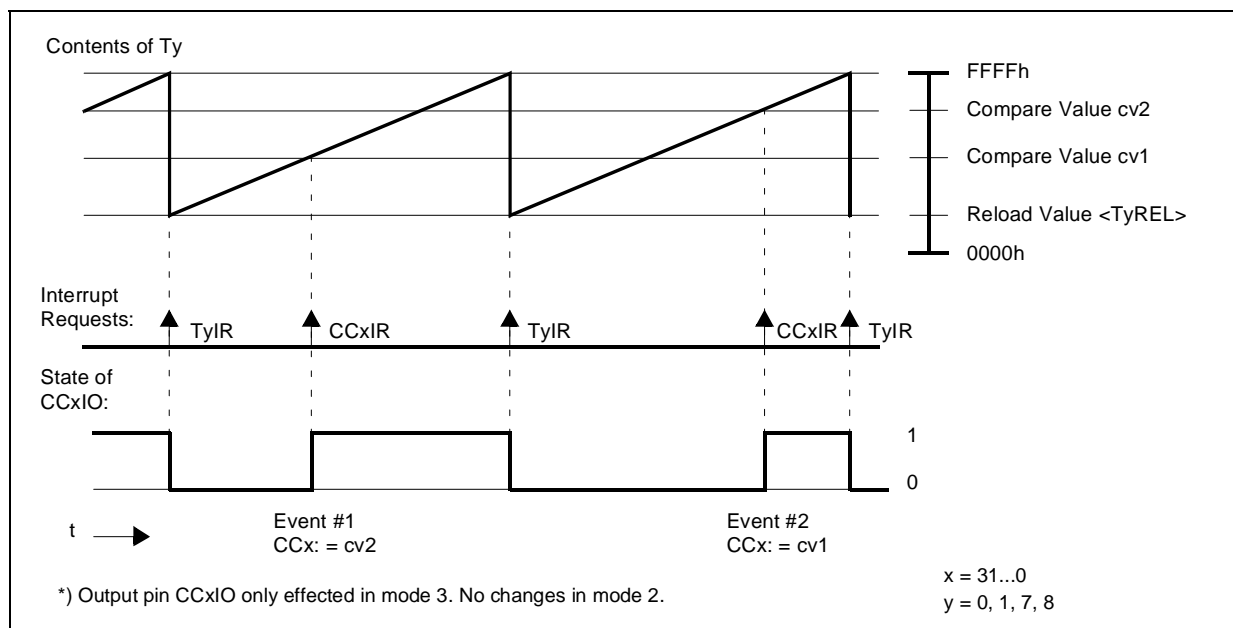
In the example below, the compare value in register CCx is modified from cv1 to cv2 after compare event #1. Compare event #2, however, will not occur until the next period of timer Ty.

**Figure 112 :** Compare mode 2 and 3 block diagram



Note The port latch and pin remain unaffected in compare mode 2.

**Figure 113 :** Timing example for compare modes 2 and 3



\*) Output pin CCxIO only effected in mode 3. No changes in mode 2.

**14.5.4 - 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 113).

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.

**14.5.5 - 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 31.

**Table 31** : 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 31) must be programmed to compare mode 1 and the corresponding bank 2 register (see Table 31) 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.

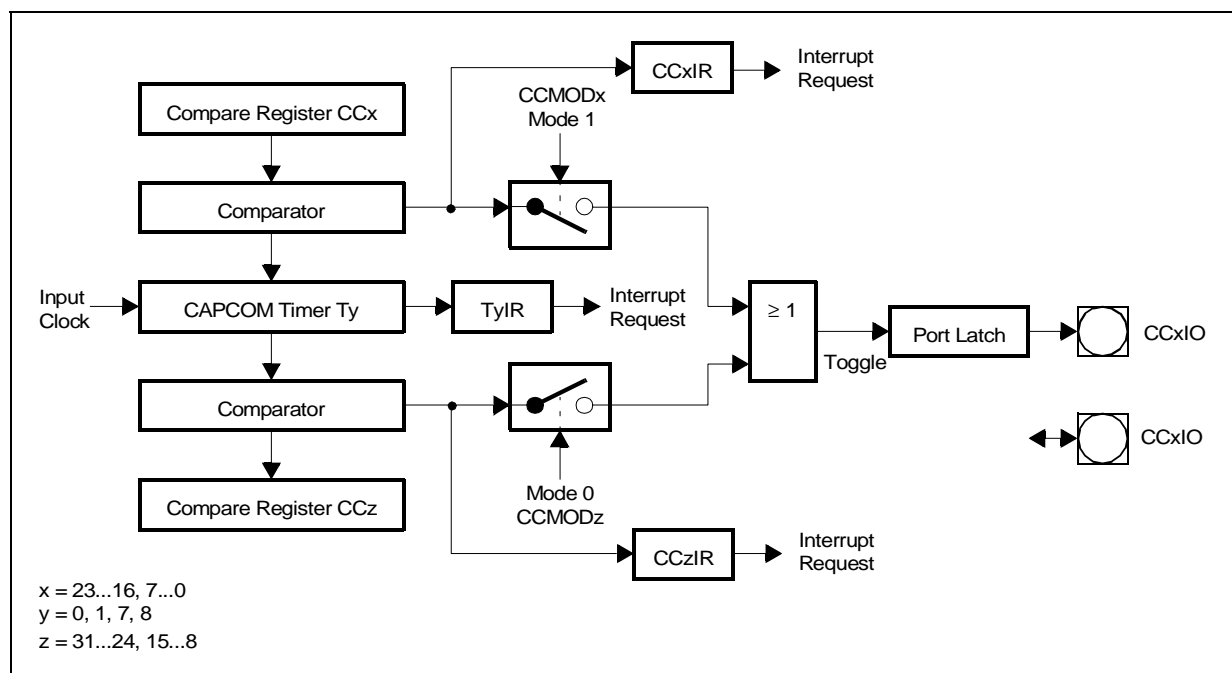
When a match is detected for one of the two registers in a register pair (CCx or CCz) the associated interrupt request flag (CCxIR or

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.

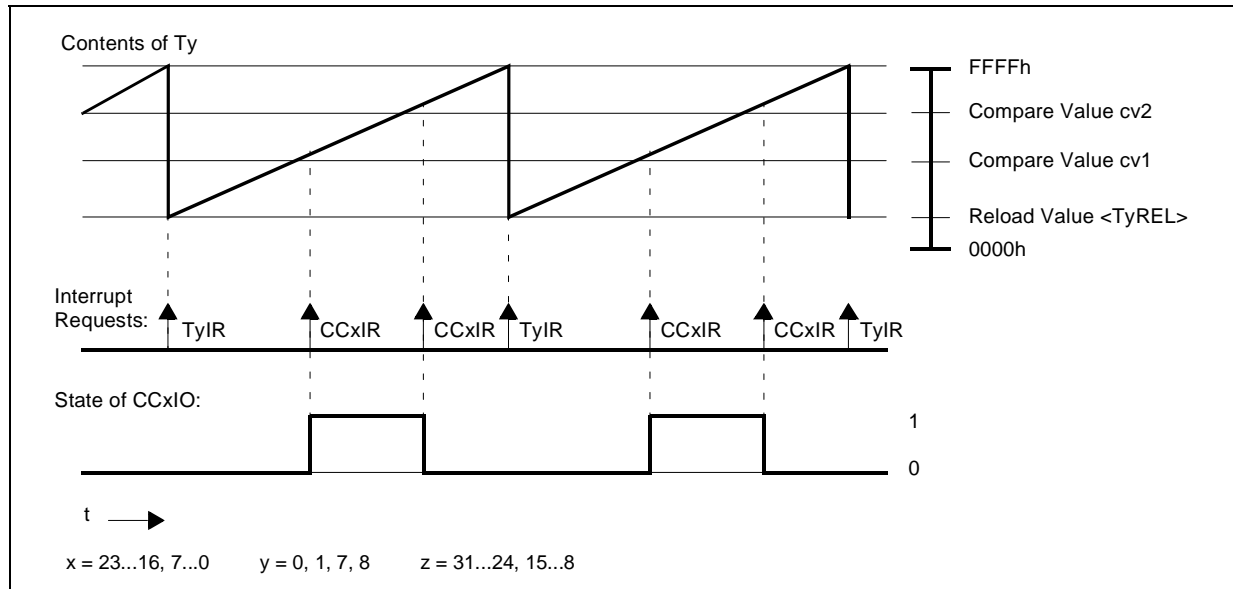
**Figure 114** : Double register compare mode block diagram



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 115 :** Timing example for double register compare mode



**14.6 - 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 Bitaddressable 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 32)								SFR/ESF				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 32 : 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

**15 - PULSE WIDTH MODULATION MODULE**

The Pulse Width Modulation (PWM) Module of the ST10X167 generates up to 4 independent PWM signals. The minimum PWM signal frequency depends on the width (16 Bit) 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 (see Figure 116 ).

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 117 ).



Figure 116 : SFRs and port pins associated with the PWM module

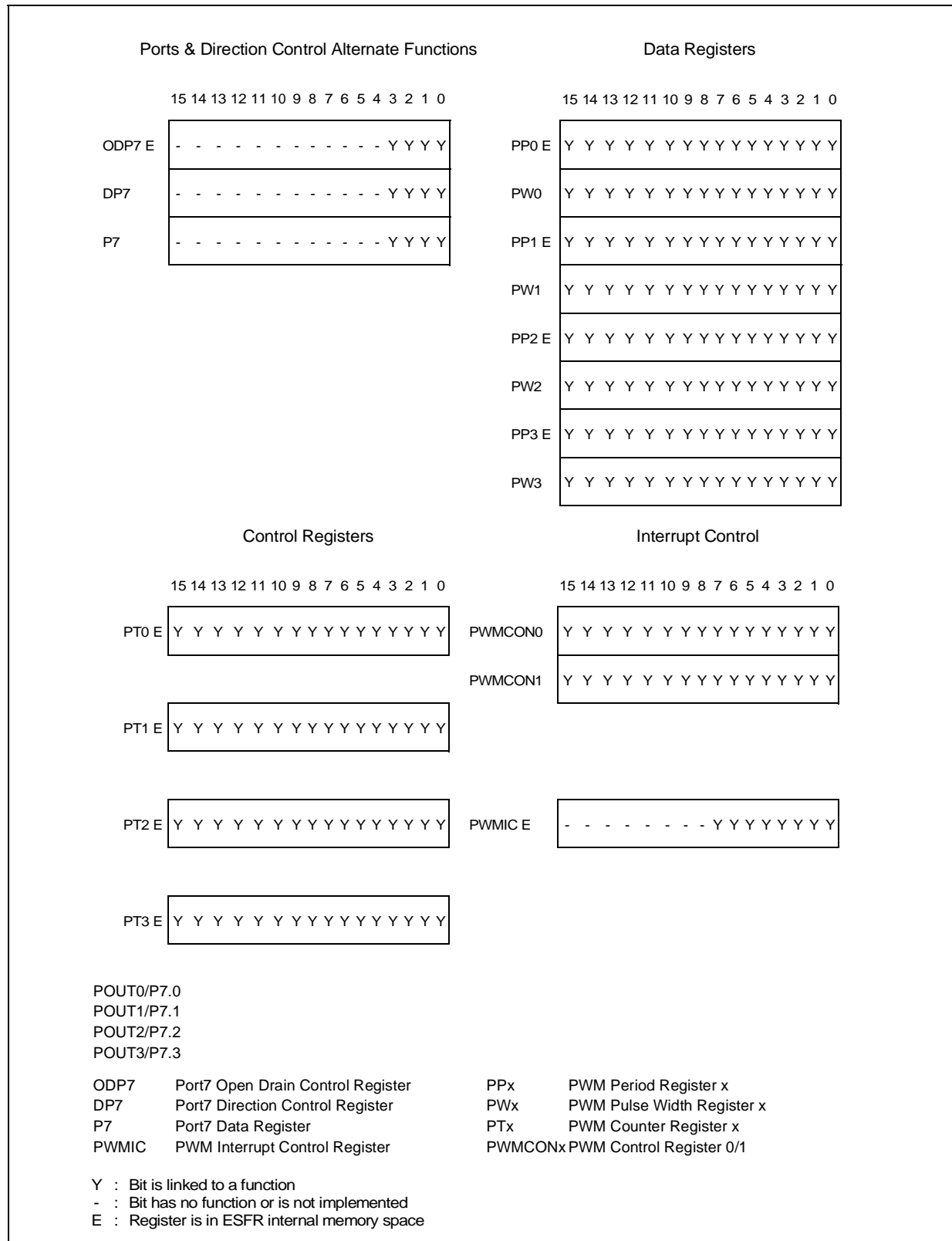
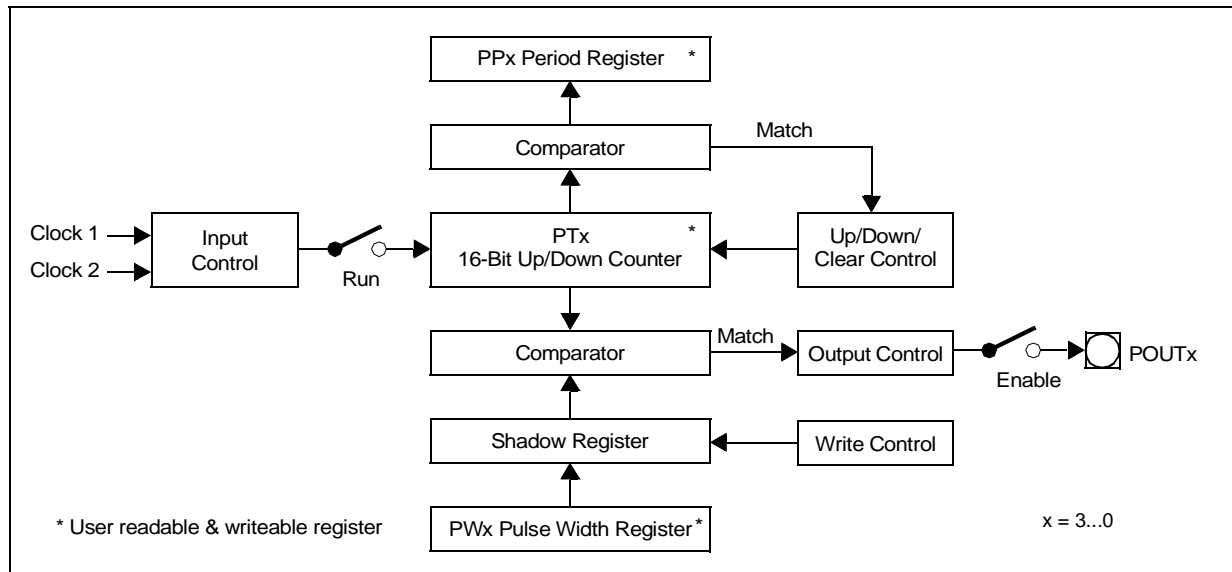


Figure 117 : PWM channel block diagram



### 15.1 - Operating Modes

The PWM module provides 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.

#### 15.1.1 - Mode 0: Standard PWM Generation (Edge Aligned PWM)

Mode 0 is selected by clearing the respective Bit PMx in register PWMCON1 to '0'. In this mode the timer PTx 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 shadow register plus 1, counted in units of the timer resolution.

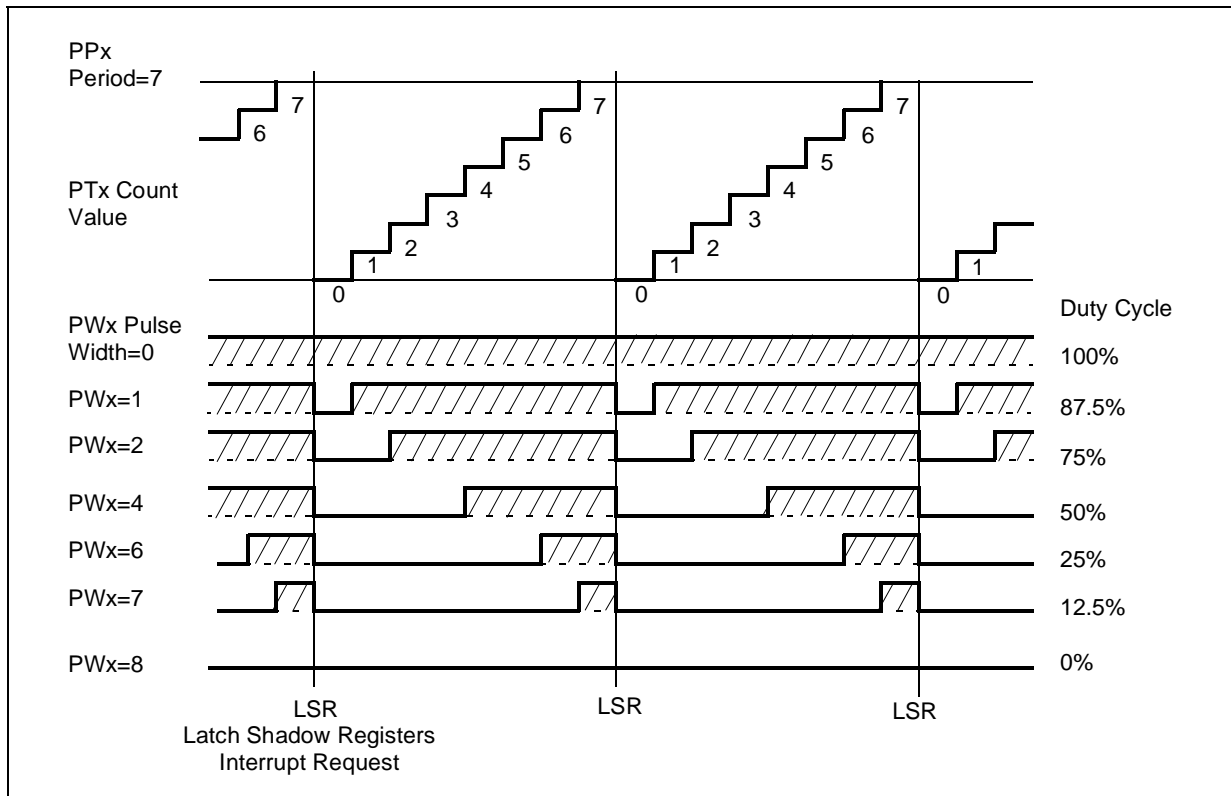
$$\text{PWM\_Period}_{\text{Mode0}} = [\text{PPx}] + 1$$

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 118 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 118 : Operation and output waveform in mode 0



**15.1.2 - Mode 1: Symmetrical PWM Generation (Center Aligned PWM)**

Mode 1 is selected by setting the respective Bit PMx in register PWMCON1 to '1'. In this mode the timer PTx 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

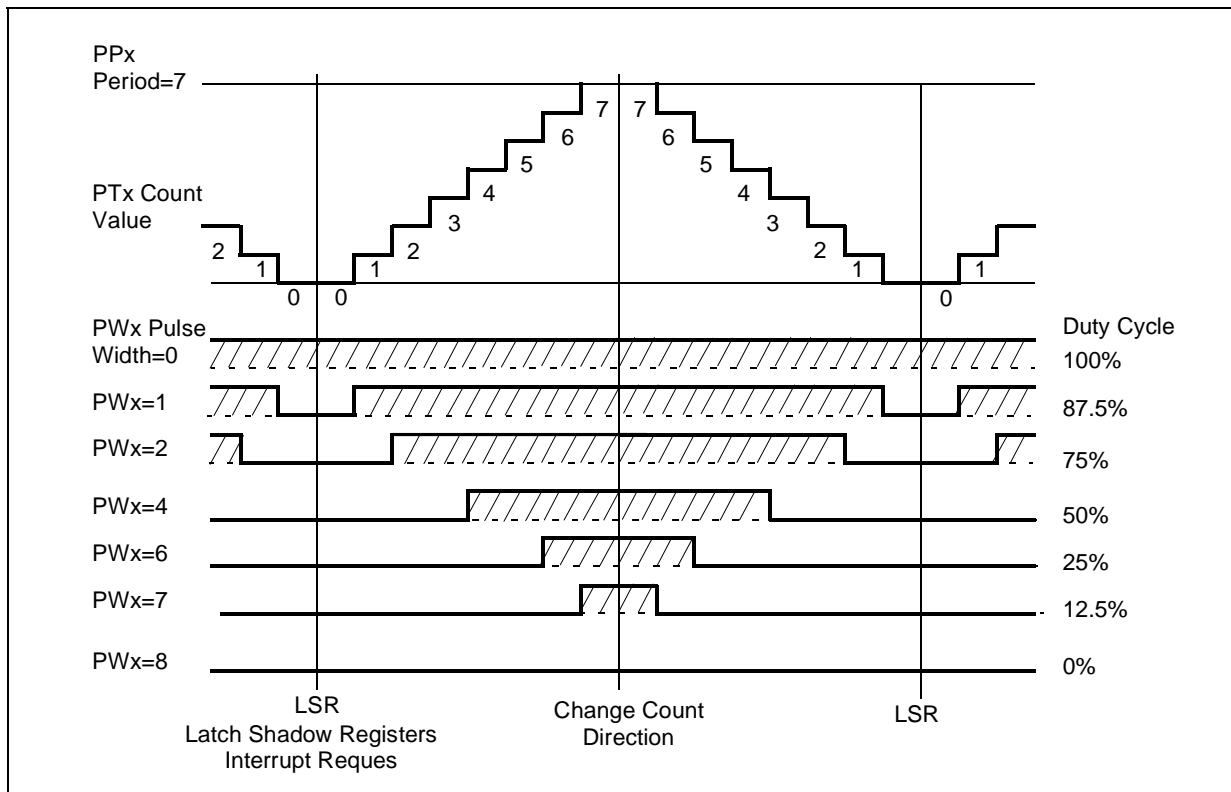
is switched back to a low level when the respective timer has counted down to a value below the contents of the pulse width shadow register. So in mode 1 this PWM value controls both edges of the output signal.

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 119 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 119** : Operation and output waveform in mode 1



### 15.1.3 - Burst Mode

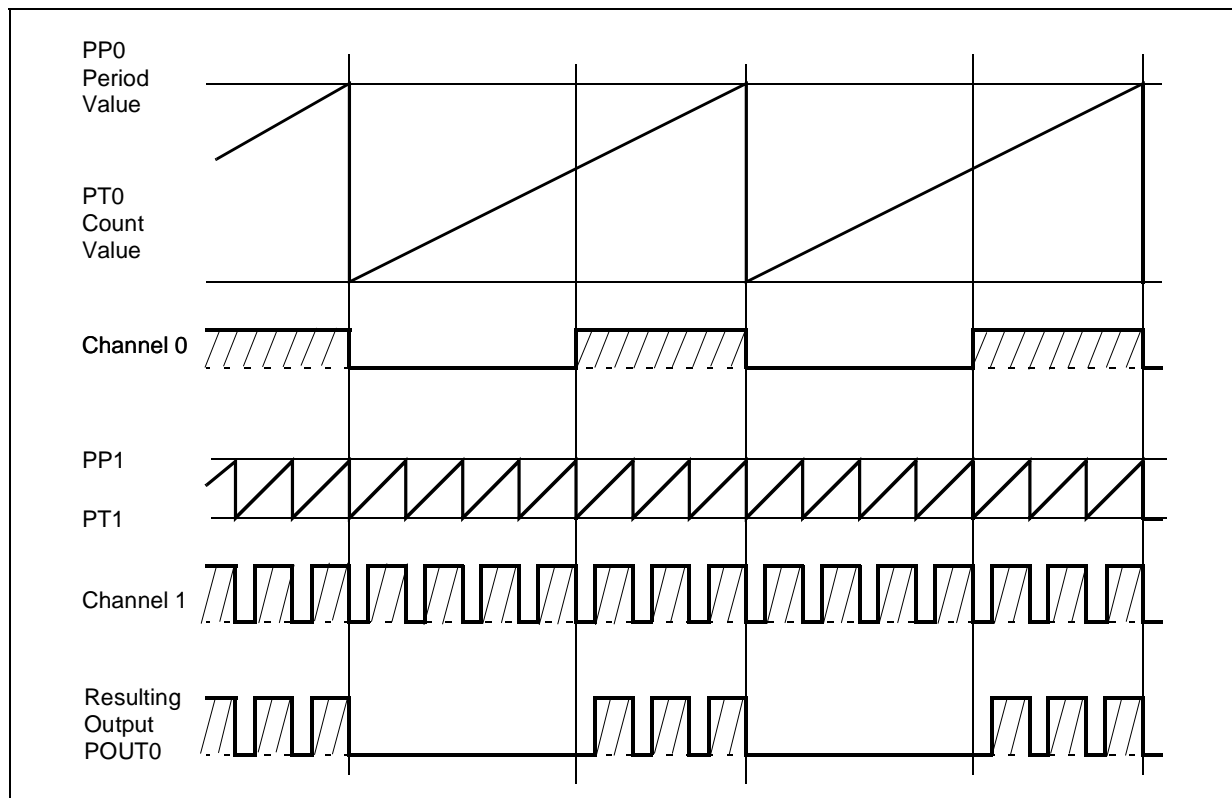
Burst mode is selected by setting Bit PB01 in register PWMCON1 to '1'. This mode combines the signals from PWM channels 0 and 1 onto the port pin of channel 0.

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. XORing of the PWM signal and the port output latch value is done after the ANDing of channel 0 and 1 (see Figure 120).

**Figure 120** : Operation and output waveform in burst mode



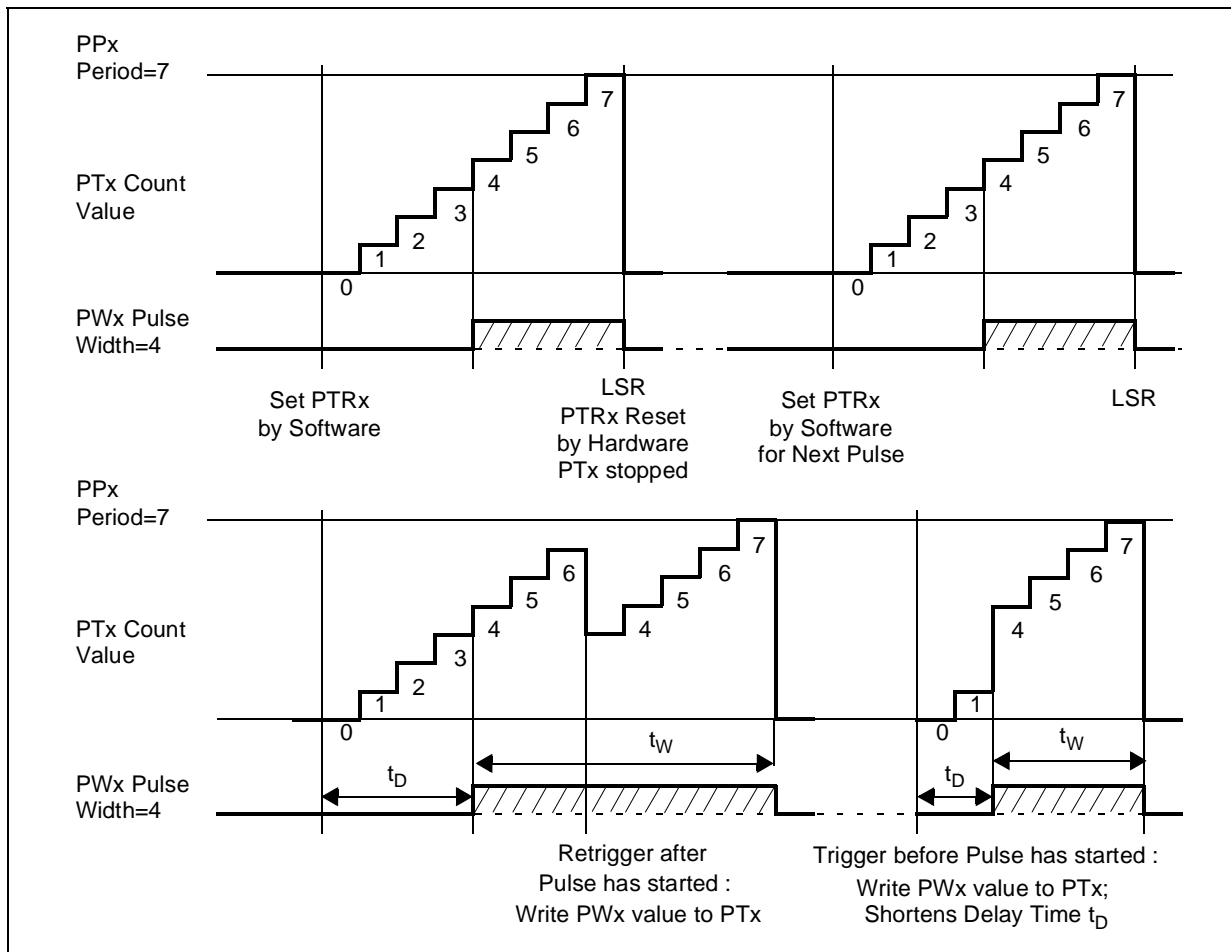
**15.1.4 - Single Shot Mode**

Single shot mode is selected by setting the respective Bit PSx in register PWMCON1 to '1'. This mode is available for PWM channels 2 and 3. In this mode the timer PTx 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 121).

**Figure 121** : 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 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 directly with the value in the respective PPx 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), the timer start value (PTx) and the pulse width value (PWx) appropriately, the pulse width (tw) and the optional pulse delay (td) may be varied in a wide range (see Figure 121).

## 15.2 - PWM Module Registers

The PWM module is controlled via two sets of registers. The waveforms are selected by the channel specific registers PTx (timer), PPx (period) and PWx (pulse width). Three common registers control the operating modes and the general functions (PWMCON0 and PWMCON1) of the PWM module as well as the interrupt behavior (PWMIC).

### Up/down Counters PTx

Each counter PTx 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 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 rather than controlling the PWM output signal.

**Note** For the register locations please refer to Table 33.

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

### Period Registers PPx

The 16 Bit period register PPx 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 register at the beginning of every new PWM cycle, or upon a write access to PPx, while the timer is stopped. The CPU accesses the PPx register while the hardware compares the contents of the shadow register with the contents of the associated counter PTx.

When a match is found between counter and PPx 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 34.

**Table 33**

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 <sub>CPU</sub> Mode 0	f <sub>cpu</sub> /2 <sup>8</sup>	f <sub>cpu</sub> /2 <sup>10</sup>	f <sub>cpu</sub> /2 <sup>12</sup>	f <sub>cpu</sub> /2 <sup>14</sup>	f <sub>cpu</sub> /2 <sup>16</sup>
f <sub>CPU</sub> / 64 Mode 0	f <sub>cpu</sub> /64x2 <sup>8</sup>	f <sub>cpu</sub> /64x2 <sup>10</sup>	f <sub>cpu</sub> /64x2 <sup>12</sup>	f <sub>cpu</sub> /64x2 <sup>14</sup>	f <sub>cpu</sub> /64x2 <sup>16</sup>
f <sub>CPU</sub> Mode 1	f <sub>cpu</sub> /2x2 <sup>8</sup>	f <sub>cpu</sub> /2x2 <sup>10</sup>	f <sub>cpu</sub> /2x2 <sup>12</sup>	f <sub>cpu</sub> /2x2 <sup>14</sup>	f <sub>cpu</sub> /2x2 <sup>16</sup>
f <sub>CPU</sub> / 64 Mode 1	f <sub>cpu</sub> /2x64x2 <sup>8</sup>	f <sub>cpu</sub> /2x64x2 <sup>10</sup>	f <sub>cpu</sub> /2x64x2 <sup>12</sup>	f <sub>cpu</sub> /2x64x2 <sup>14</sup>	f <sub>cpu</sub> /2x64x2 <sup>16</sup>

**Pulse Width Registers PWx**

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 register while the hardware compares the contents of the shadow register with the contents of the associated counter PTx. The shadow register is loaded from the respective PWx register at the beginning of

every new PWM cycle, or upon a write access to PWx, 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:

$$\text{PWM output signal} = [\text{PTx}] \geq [\text{PWx shadow latch}]$$

This type of comparison allows a flexible control of the PWM signal. For the register locations refer to the Table 34.

**Table 34 :** 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

**PWM Control Register PWMCON0**

Register PWMCON0 controls the function of the timers of the four PWM channels and the channel specific interrupts. Having the control Bit organized in functional groups allows to start or to stop all the 4 PWM timers simultaneously with one Bitfield instruction.

**PWMCON0 (FF30h / 98h)**

SFR

Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>PIR3</b>	<b>PIR2</b>	<b>PIR1</b>	<b>PIR0</b>	<b>PIE3</b>	<b>PIE2</b>	<b>PIE1</b>	<b>PIE0</b>	<b>PTI3</b>	<b>PTI2</b>	<b>PTI1</b>	<b>PTI0</b>	<b>PTR3</b>	<b>PTR2</b>	<b>PTR1</b>	<b>PTR0</b>
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
PTRx	<b>PWM Timer x Run Control Bit</b> '0': Timer PTx is disconnected from its input clock '1': Timer PTx is running
PTIx	<b>PWM Timer x Input Clock Selection</b> '0': Timer PTx clocked with CLK <sub>CPU</sub> '1': Timer PTx clocked with CLK <sub>CPU</sub> / 64
PIEx	<b>PWM Channel x Interrupt Enable Flag</b> '0': Interrupt from channel x disabled '1': Interrupt from channel x enabled
PIRx	<b>PWM Channel x Interrupt Request Flag</b> '0': No interrupt request from channel x '1': Channel x interrupt pending (must be reset via software)



**PWM Control Register PWMCON1**

Register PWMCON1 controls the operating modes and the outputs of the four PWM channels. 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 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

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	<b>PWM Channel x Output Enable Bit</b> '0': Channel x output signal disabled, generate interrupt only '1': Channel x output signal enabled
PMx	<b>PWM Channel x Mode Control Bit</b> '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	<b>PWM Channel 0/1 Burst Mode Control Bit</b> '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	<b>PWM Channel x Single Shot Mode Control Bit</b> '0': Channel x works in respective standard mode '1': Channel x operates in single shot mode

**15.3 - Interrupt Request Generation**

Each of the four channels of the PWM 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. The interrupt service routine can determine the active channel interrupt(s) from the channel specific interrupt request flags PIRx in register PWMCON0.

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 and PWx are now ready to receive a new value. If a channel interrupt is enabled via its

respective PIRx Bit, also the common interrupt request flag PWMIR in register PWMIC is set, provided that it is enabled via the common interrupt enable Bit PWMIE.

Note The channel interrupt request flags (PIRx in register PWMCON0) 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 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

Note Refer to the general Interrupt Control Register description for an explanation of the control fields.

**15.4 - PWM Output Signals**

The output signals of the four PWM channels (POUT3...POUT0) are alternate output functions on Port7 (P7.3...P7.0). The output signal of each PWM channel is individually enabled by control Bit PENx in register PWMCON1.

The PWM signals are XORed with the respective port latch outputs before being driven to the port pins.

This allows driving the PWM signal directly to the port pin (P7.x='0') or drive the inverted PWM signal (P7.x='1') (see Figure 122).

**Note** Using the open drain mode on Port 7 allows the combination of two or more PWM outputs through a Wired-AND configuration, using an external pull-up device. This provides sort of a burst mode for any PWM channel.

**Software Control of the PWM 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:

$$\text{PWM output signal} = [\text{PTx}] \geq [\text{PWx shadow latch}]$$

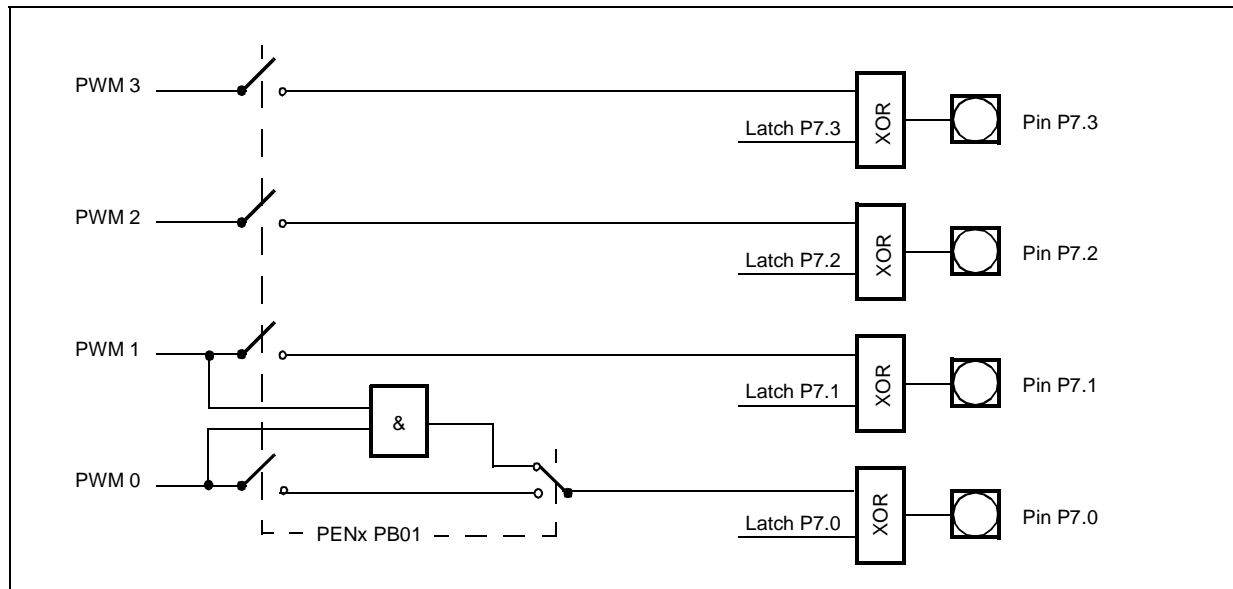
So whenever software changes registers PTx, the respective output will reflect the condition after the change. Loading timer PTx with a value greater than or equal to the value in PWx immediately sets the respective output, a PTx value below the PWx value clears the respective output.

By clearing or setting the respective Port7 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 122 : PWM output signal generation**



### 16 - ANALOG / DIGITAL CONVERTER

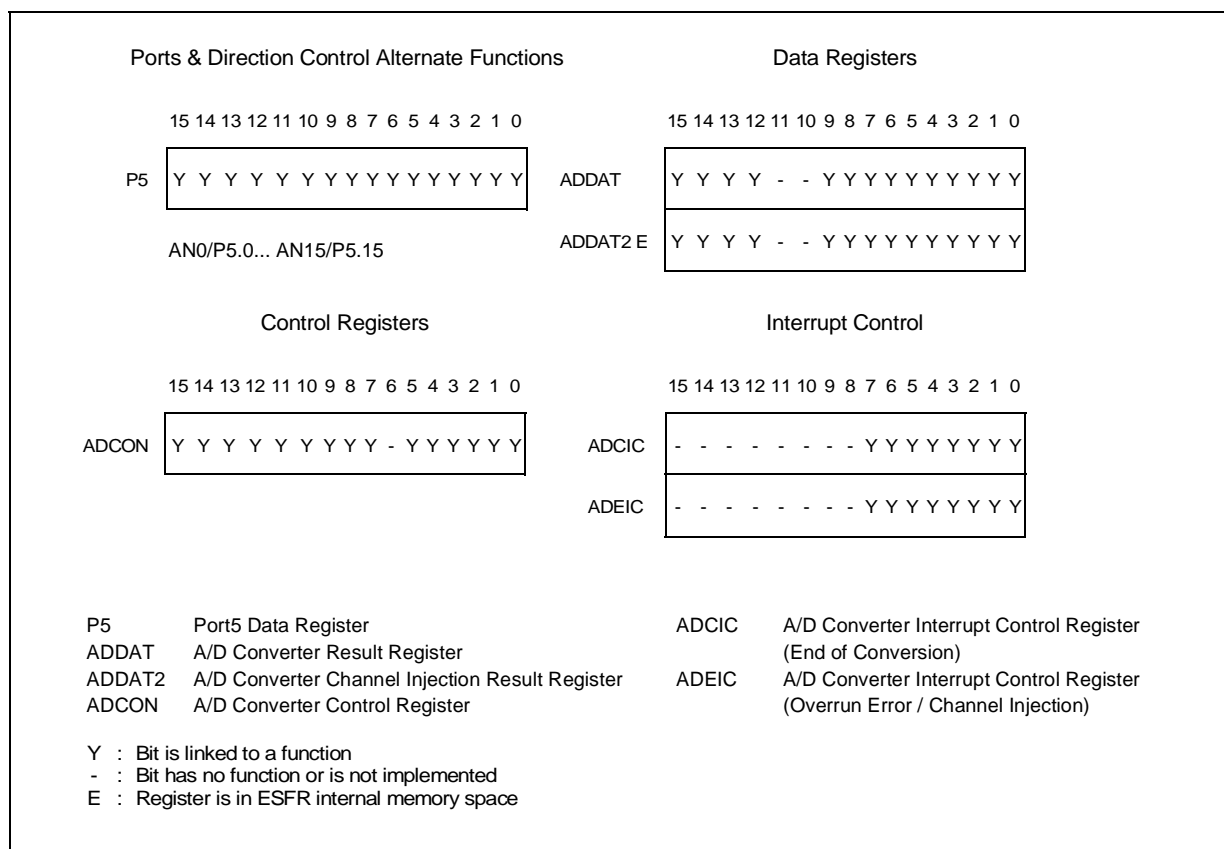
The ST10X167 provides an Analog / Digital Converter with 10 Bit resolution and a sample & hold circuit on-chip. A multiplexer selects between up to 16 analog input channels (alternate functions of Port5) either via software (fixed channel modes) or automatically (auto scan modes). An automatic self-calibration adjusts the ADC module to changing temperatures or process variations. 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.

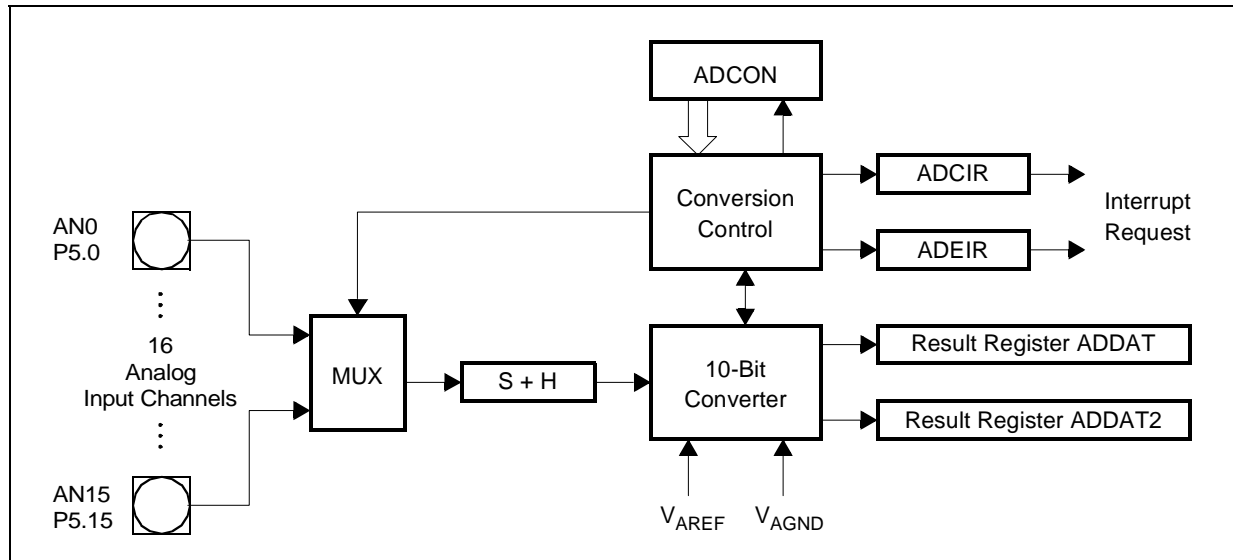
Figure 123 : SFRs and port pins associated with the A/D converter



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 124** : Analog / digital converter block diagram



**16.1 - 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	<b>ADC Analog Channel Input Selection</b>
ADM	<b>ADC Mode Selection</b> 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	<b>ADC Start Bit</b>
ADBSY	<b>ADC Busy Flag</b> ADBSY = 1: a conversion is active
ADWR	<b>ADC Wait for Read Control</b>
ADCIN	<b>ADC Channel Injection Enable</b>
ADCRQ	<b>ADC Channel Injection Request Flag</b>
ADSTC	<b>ADC Sample Time Control *)</b>
ADCTC	<b>ADC Conversion Time Control *)</b>

Note \*) ADSTC and ADCTC control the conversion timing. Refer to Section 16.2 Conversion Timing Control.

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	<b>A/D Conversion Result (10 Bit)</b>
CHNR	<b>Channel Number (4 Bit, identifies the converted analog channel)</b>

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).

#### **16.1.1 - 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.

#### **16.1.2 - Auto Scan Conversion Modes**

These modes are selected by programming the mode selection field ADM in register ADCON to '10<sub>B</sub>' (single conversion) or to '11<sub>B</sub>' (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.

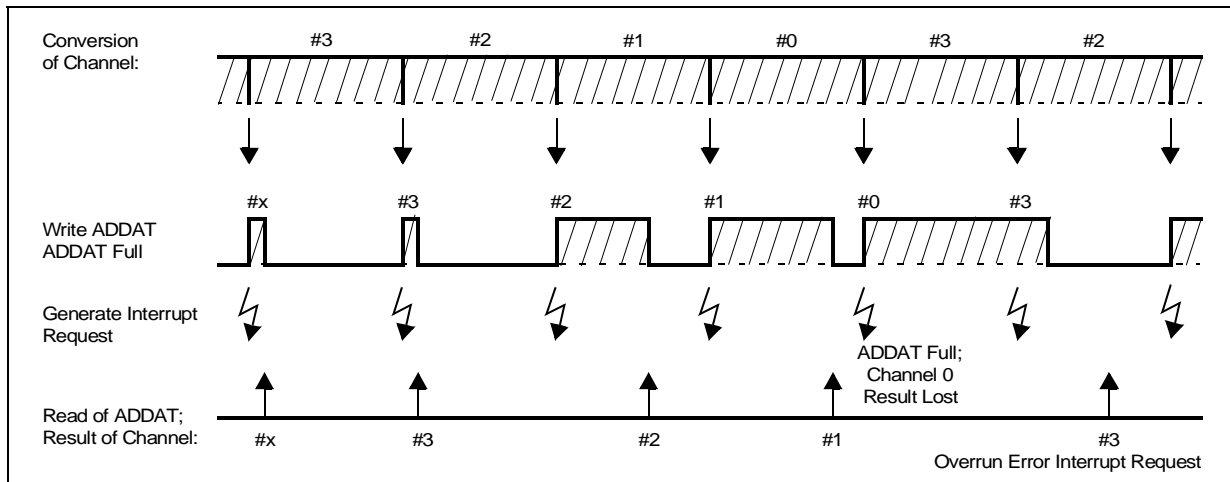
After the conversion is complete, the interrupt request flag ADCIR will be set and the converter will automatically start a new conversion of the next lower channel. ADCIR will be set after each completed conversion. After conversion of channel 0 the current sequence is complete.

**In single conversion mode** the converter will automatically stop and reset Bit 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 125 : Auto scan conversion mode example



16.1.3 - 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.

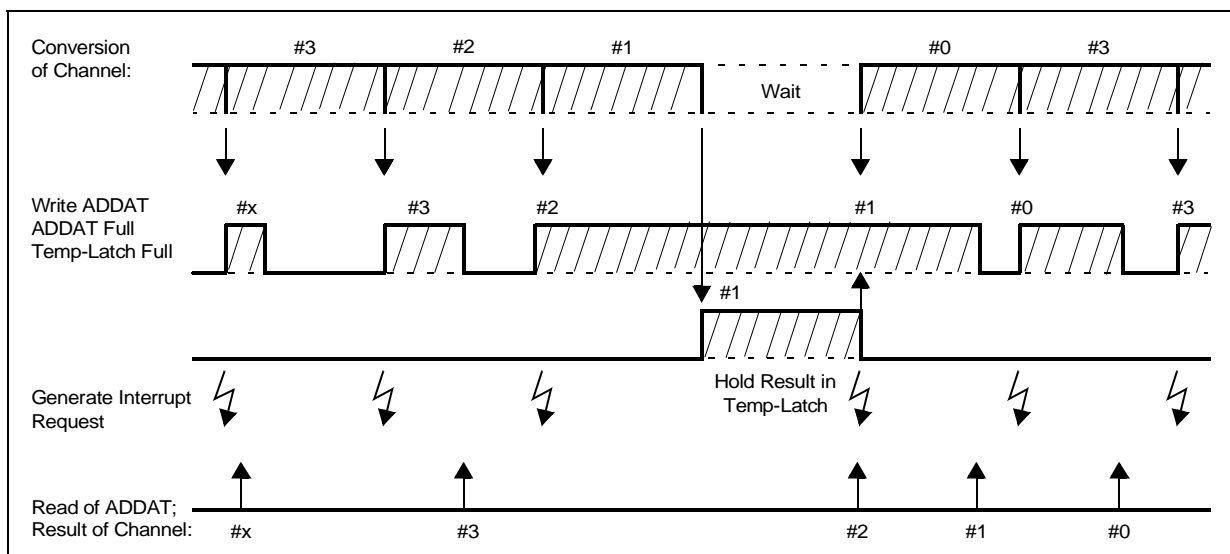
In order to avoid error interrupts and the loss of conversion results especially when using continuous conversion modes, the ADC can be switched to "Wait for ADDAT Read Mode" by setting Bit ADWR in register ADCON.

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 126 : Wait for read mode example



**16.1.4 - 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 Bit 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, Bitfield 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 127).

A channel injection can be triggered in two ways:

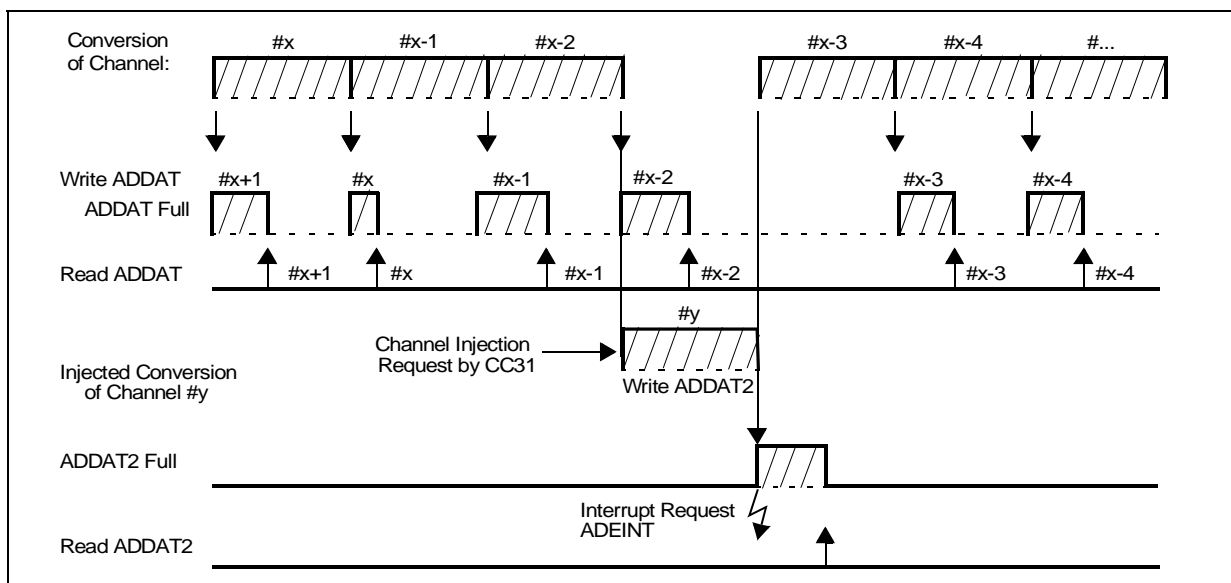
- Setting the Channel Injection Request Bit ADCRQ via software a compare or a capture event of Capture/Compare register CC31 of the CAPCOM2 Unit, which also sets Bit ADCRQ.
- Triggering a channel injection at a specific time on the occurrence of a predefined count value of the CAPCOM timers or on a capture 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).

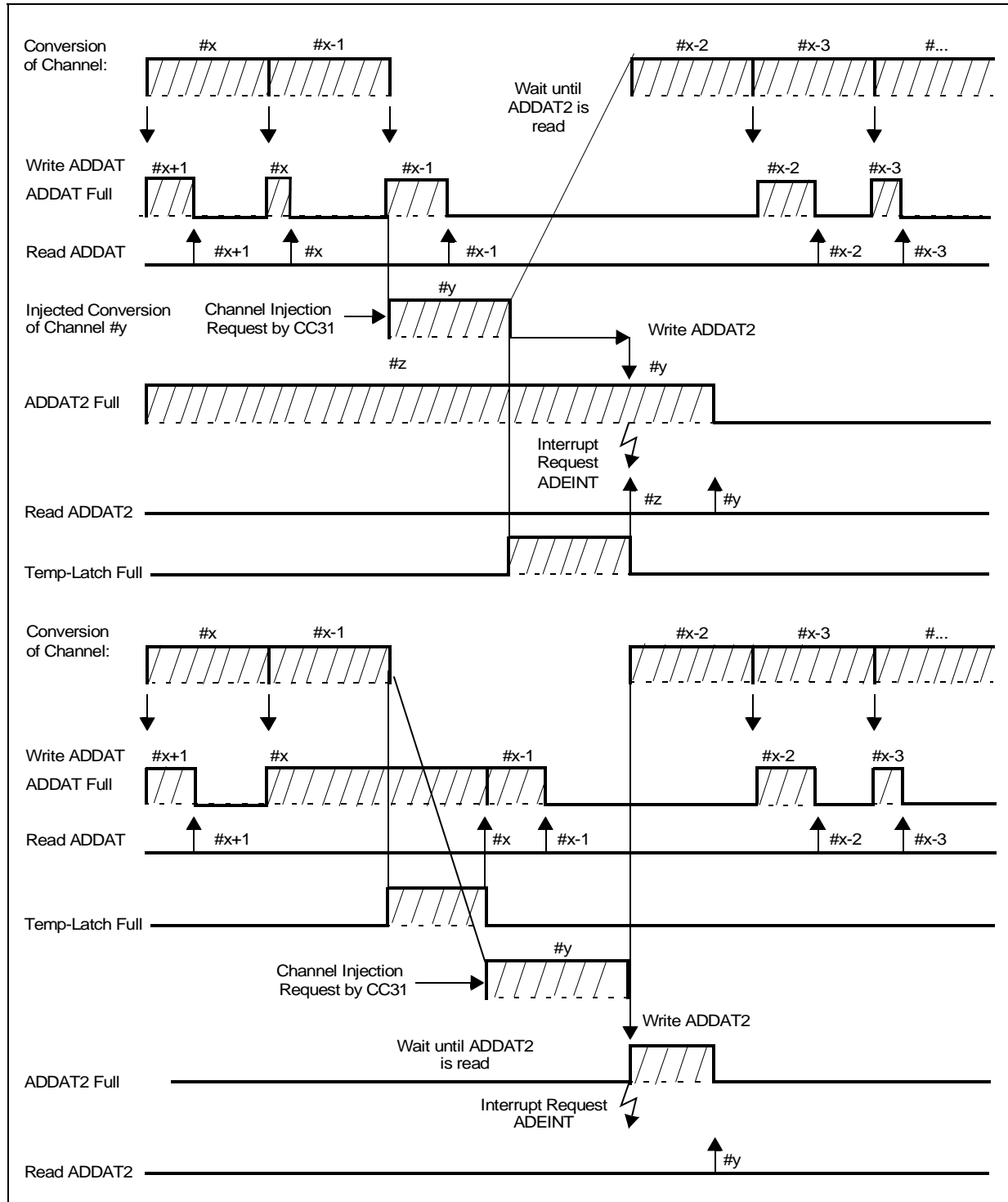
**Figure 127 : Channel injection example**





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 128** : Channel injection example with wait for read



## 16.2 - Conversion Timing Control

When a conversion is started, first the capacitances of the converter are loaded via the respective analog input pin to the current analog input voltage. The time to load the capacitances is referred to as sample time. Next the sampled voltage is converted to a digital value in 10 successive steps, which correspond to the 10 Bit resolution of the ADC. The next 4 steps are used for an internal self-calibration of the converter module. During these 14 steps the internal capacitances are repeatedly charged and discharged via the  $V_{AREF}$  pin.

The current that has to be drawn from the sources for sampling and changing charges depends on the time that each respective step takes, because the capacitors must reach their final voltage level within the given time, at least with a certain approximation. The maximum current, however, that a source can deliver, depends on its internal resistance.

The time that the two different actions during conversion take (sampling, and converting) can be programmed within a certain range in the ST10X167 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 ST10X167 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  $f_{CPU} = 1/2TCL$ .

ADCTC	Conversion clock $t_{CC}$	ADSTC	Sample clock $t_{SC}$
00	TCL x 24	00	$t_{CC}$
01	Reserved, do not use	01	$t_{CC} \times 2$
10	TCL x 96	10	$t_{CC} \times 4$
11	TCL x 48	11	$t_{CC} \times 8$

A complete conversion will take  $14t_{CC} + 2t_{SC} + 4TCL$ . This time includes the conversion itself, the sample time and the time required to transfer the digital value to the result register.

Note The decoding of Bit field ADCTC provides compatibility with ST10F166 designs for the default value (00 at after reset).

## 16.3 - Calibration

**A full calibration sequence is performed after a reset.** This full calibration lasts 40 000 CPU clock cycles. During this time, the busy flag ADBSY is set to indicate the operation. Normal conversion may be performed during this time. The duration of the calibration sequence is then extended by the time consumed by the conversions.

Note: After a power-on reset, the total unadjusted error (TUE) of the ADC might be worse than  $\pm 2$  LSB (max.  $\pm 4$  LSB). During the full calibration sequence, the TUE is constantly improved until at the end of the calibration, TUE is within the specified limits of  $\pm 2$  LSB.

**One calibration cycle is performed after each conversion.** Each calibration cycle takes 4 ADC clock cycles. These operation cycles ensure constant updating of the ADC's accuracy, compensating for changing operating conditions. A complete conversion cycle takes 16 ADC clock cycles (2 ADC clocks sample phase, 10 ADC clocks conversion phase, 4 ADC clocks calibration phase).

## 16.4 - 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 5.1.3 - Interrupt Control Registers for explanation of the control fields.

### 17 - ON-CHIP CAN INTERFACE

The CAN module is always enabled on the ST10F167. The CAN module may be enabled or disabled with ST10C167 or ST10R167 by programming the XPEN Bit in the SYSCON register.

The integrated CAN Module handles the autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active), the on-chip CAN Module can receive and transmit standard frames with 11 Bit identifiers and extended frames with 29 Bit identifiers.

The CAN provides **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 Byte.

The Bit timing is derived from the XCLK and is programmable up to a data rate of 1M Baud (at 25 MHz CPU clock). The CAN Module uses two pins of Port 4 to interface to a bus transceiver.

#### 17.1 - The CAN Controller

The CAN module combines 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.

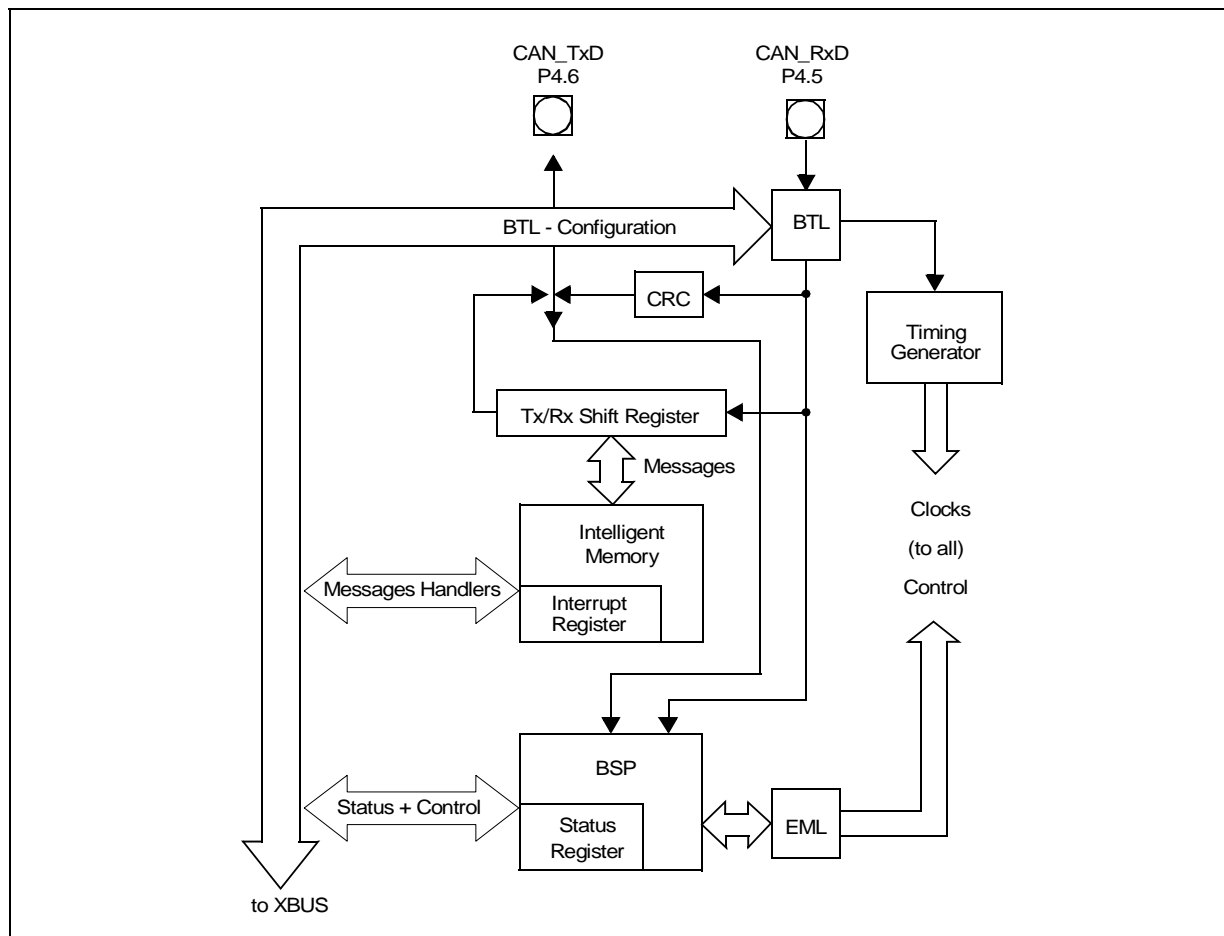
By requesting the transmission of a message with the direction set as receive, a remote frame can be sent to request that the appropriate object be sent by some other node.

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 Bit and one for identifiers of 29 Bit. 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 129 : 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 busline input CAN\_RxD and handles the busline related Bit timing according to the CAN protocol.

The BTL synchronizes on a recessive to dominant busline transition at Start of Frame (hard synchronization) and on any further recessive to dominant busline 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

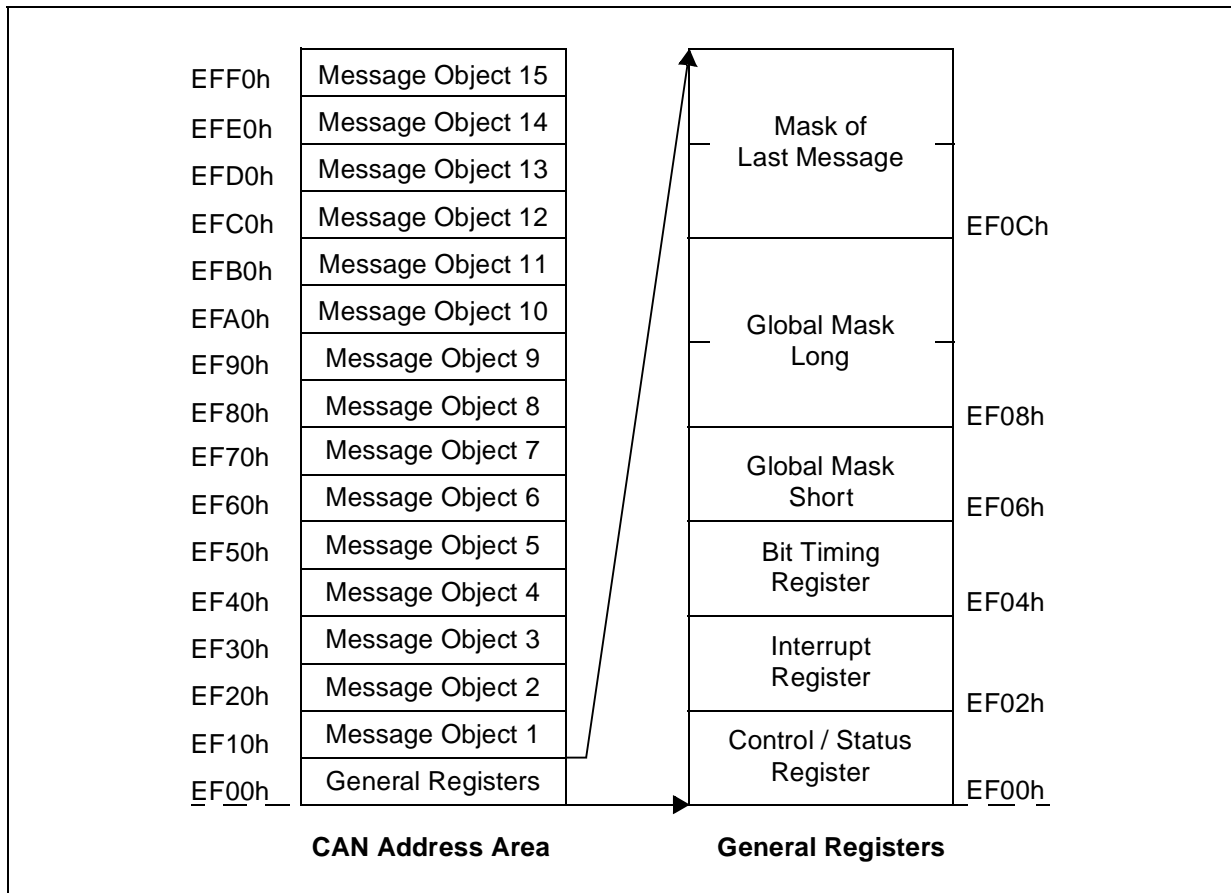
maximum 8 data Byte length. Each of these objects has a unique identifier and its own set of control and status Bit. After the initial configuration, the Intelligent Memory can handle the reception and transmission of data without further CPU actions.

### 17.2 - Register and Message Object Organization

All registers and message objects of the CAN controller are located in the special CAN address area of 256 Byte, which is mapped into segment 0 and uses addresses 00'EF00h through 00'EFFh. 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 130 lists the registers which are part of the CAN controller. There are also ST10X167 specific registers that are associated with the CAN Module. These registers, however, control the access to the CAN Module rather than its function.

Figure 130 : CAN module address map



**Control / Status Register (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 35 : CAN Control/Status register

Bit	Function (Control Bit)
INIT	<p><b>Initialization</b></p> <p>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 139.</p> <p>0: Disable software initialization of the CAN controller; on INI completion, the CAN waits for 11 consecutive recessive Bit before taking part in bus activities.</p>
IE	<p><b>Interrupt Enable</b> - Does not affect status updates.</p> <p>1: Global interrupt enable from CAN module.</p> <p>0: Global interrupt disable from CAN module.</p>
SIE	<p><b>Status Change Interrupt Enable</b></p> <p>1: Enables interrupt generation when a message transfer (reception or transmission is successfully completed) or CAN bus error is detected and registered in LEC is the status partition.</p> <p>0: Disable status change interrupt.</p>
EIE	<p><b>Error Interrupt Enable</b></p> <p>1: Enables interrupt generation on a change of Bit BOFF or EWARN in the status partition.</p> <p>0: Disable error interrupt.</p>
CCE	<p><b>Configuration Change Enable</b></p> <p>1: Allows CPU access to the Bit timing register</p> <p>0: Disables CPU access to the Bit timing register</p>
TST	<p><b>Test Mode (Bit 7)</b></p> <p>Make sure that Bit 7 is cleared when writing to the Control Register. Writing a 1 during normal operation may lead erroneous device behaviour.</p>
LEC	<p><b>Last Error Code</b></p> <p>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.</p> <p>0: <b>No Error</b></p> <p>1: <b>Stuff Error:</b> More than 5 equal Bit in a sequence have occurred in a part of a received message where this is not allowed.</p> <p>2: <b>Form Error:</b> A fixed format part of a received frame has the wrong format.</p> <p>3: <b>AckError:</b> The message this CAN controller transmitted was not acknowledged by another node</p> <p>4: <b>Bit1Error:</b> During the transmission of a message (with the exception of the arbitration field), the device wanted to send a <i>recessive</i> level ("1"), but the monitored bus value was <i>dominant</i></p> <p>5: <b>Bit0Error:</b> During the transmission of a message (or acknowledge Bit, active error flag, or overload flag), the device wanted to send a <i>dominant</i> level ("0"), but the monitored bus value was <i>recessive</i>. During <i>busoff</i> recovery this status is set each time a sequence of 11 <i>recessive</i> Bit has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at <i>dominant</i> or continuously disturbed).</p> <p>6: <b>CRCErrror:</b> The CRC check sum was incorrect in the message received.</p>
TXOK	<p><b>Transmitted Message Successfully</b></p> <p>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!).</p>



Bit	Function (Control Bit)
RXOK	<b>Received Message Successfully</b> 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	<b>Error Warning Status</b> Indicates that at least one of the error counters in the EML has reached the error warning limit of 96.
BOFF	<b>Busoff Status</b> 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.

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.

### 17.3 - CAN Interrupt Handling

The on-chip CAN Module has one interrupt output, which is connected (through a synchronization stage) to a standard interrupt node in the ST10X167 in the same manner as all other interrupts of the standard on-chip peripherals. The control register for this 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). 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). As for all other interrupts, the interrupt request flag XP0IR in register XP0IC is cleared automatically by hardware when this interrupt is serviced (either by standard interrupt or PEC service).

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).

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

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 36 lists the valid values for INTID and their corresponding interrupt sources.



Bit	Function
INTID	<b>Interrupt Identifier</b> This number indicates the cause of the interrupt. When no interrupt is pending, the value will be "00".

**Table 36 : INTID values and Corresponding Interrupt Sources**

INTID	Cause of the Interrupt
00	<b>Interrupt Idle:</b> There is no interrupt request pending.
01	<b>Status Change Interrupt:</b> 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	<b>Message 15 Interrupt:</b> 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. <sup>1)</sup>
(2+N)	<b>Message N Interrupt:</b> Bit INTPND in the Message Control Register of message object 'N' has been set (N = 1...14). <sup>1) 2)</sup>

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.

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).

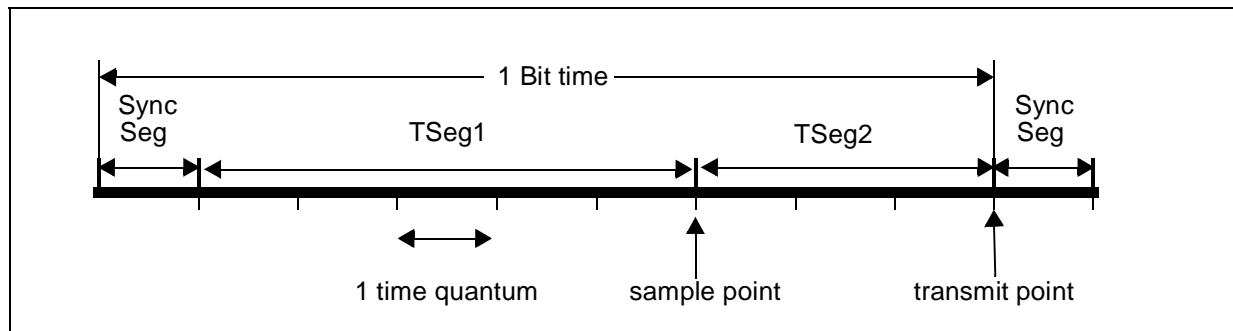
Note For exact definition of these segments please refer to the CAN Specification.

**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.

**Figure 131 : Bit timing definition**



**Bit Timing Register (EF04h)**

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	<b>Baud Rate Prescaler</b> For generating the Bit time quanta the CPU frequency is divided by 2 x (BRP+1).
SJW	<b>(Re)Synchronization Jump Width</b> Adjust the Bit time by maximum (SJW+1) time quanta for re-synchronization.
TSEG1	<b>Time Segment before sample point</b> There are (TSEG1+1) time quanta before the sample point. Valid values for TSEG1 are "2...15".
TSEG2	<b>Time Segment after sample point</b> 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 Bit 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)**

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	<b>Identifier (11 Bit)</b> Mask to filter incoming messages with standard identifier.

**Upper Global Mask Long (EF08h)**

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							

## ST10X167

**Lower Global Mask Long (EF0Ah)** 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	<b>Identifier (29 Bit)</b> Mask to filter incoming messages with extended identifier.

**Upper Mask of Last Message (EF0Ch)** 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)** 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	<b>Identifier (29 Bit)</b> Mask to filter the last incoming message (Nr. 15) with standard or extended identifier (as configured).

### 17.4 - 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 Byte (see Figure 132) and starts at an address that is a multiple of 16.

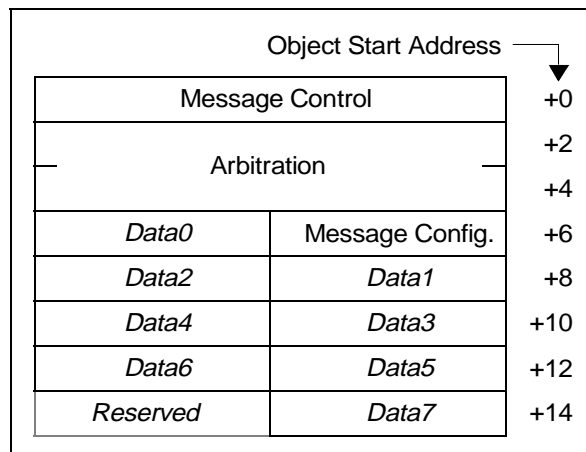
**Note** All message objects must be initialized by the CPU, even those which are not going to be used, before clearing the INIT Bit..

Each element of the Message Control Register is made of two complementary Bit.

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 37 shows how to use and to interpret these 2 Bit fields.

**Figure 132 : Message object address map**



**Table 37 : Functions of complementary Bit 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)								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	<b>Interrupt Pending</b> 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	<b>Receive Interrupt Enable</b> Defines, if Bit INTPND is set after successful reception of a frame.
TXIE	<b>Transmit Interrupt Enable</b> Defines, if Bit INTPND is set after successful transmission of a frame. <sup>1</sup>
MSGVAL	<b>Message Valid</b> 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	<b>New Data</b> 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. <sup>2</sup>
MSGLST (Receive)	<b>Message Lost</b> (This Bit applies to <i>receive</i> -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.
CPUUPD (Transmit)	<b>CPU Update</b> (This Bit applies to <i>transmit</i> -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	<b>Transmit Request</b> 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. <sup>1 3</sup>
RMTPND	<b>Remote Pending</b> (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 Bit 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.

**17.5 - 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)										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)										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	<b>Identifier (29 Bit)</b> 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)								XReg	ResetValue:--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	<b>Extended Identifier</b> Indicates, if this message object will use an extended 29 Bit identifier or a standard 11 Bit identifier.
DIR	<b>Message Direction</b> DIR="1": transmit. On TXRQ, the respective message object is transmitted. On reception of a remote frame with matching identifier, the TXRQ and RMTDND 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	<b>Data Length Code</b> 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 (locations 00'EFnFh is 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 133** : CAN controller handling of message objects in transmit direction

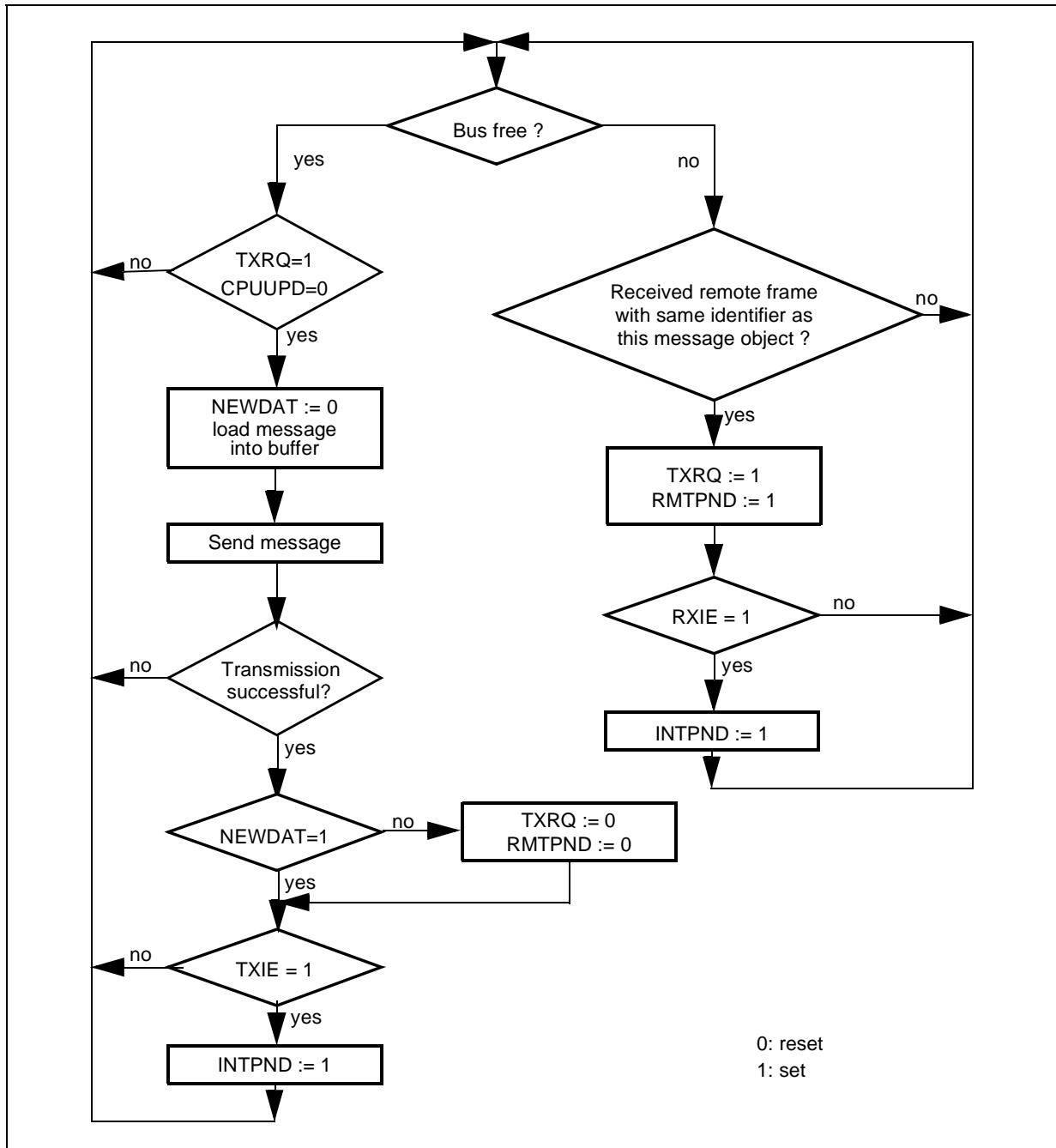




Figure 134 : CPU handling of message objects in transmit direction

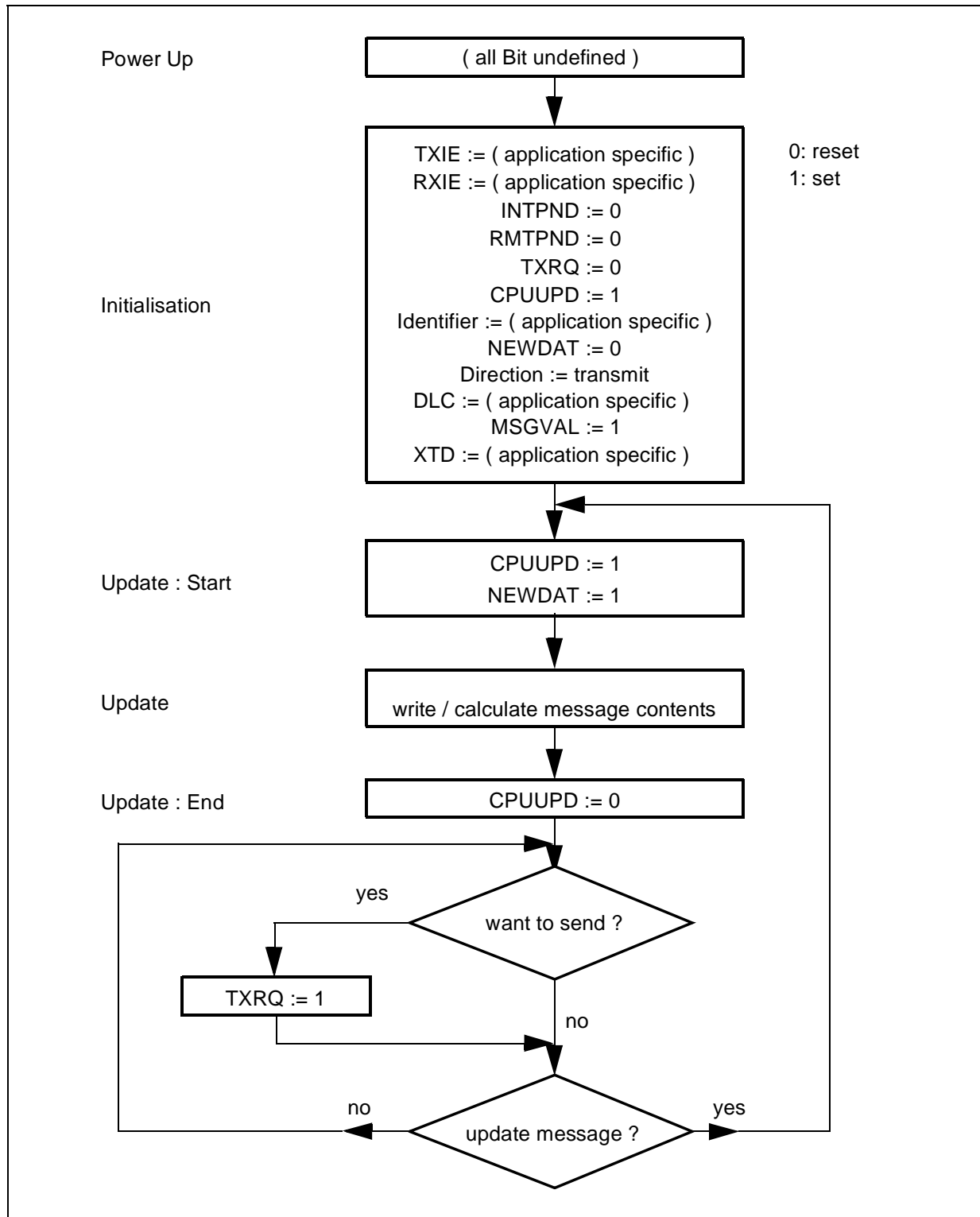


Figure 135 : CAN controller handling of message objects in receive direction

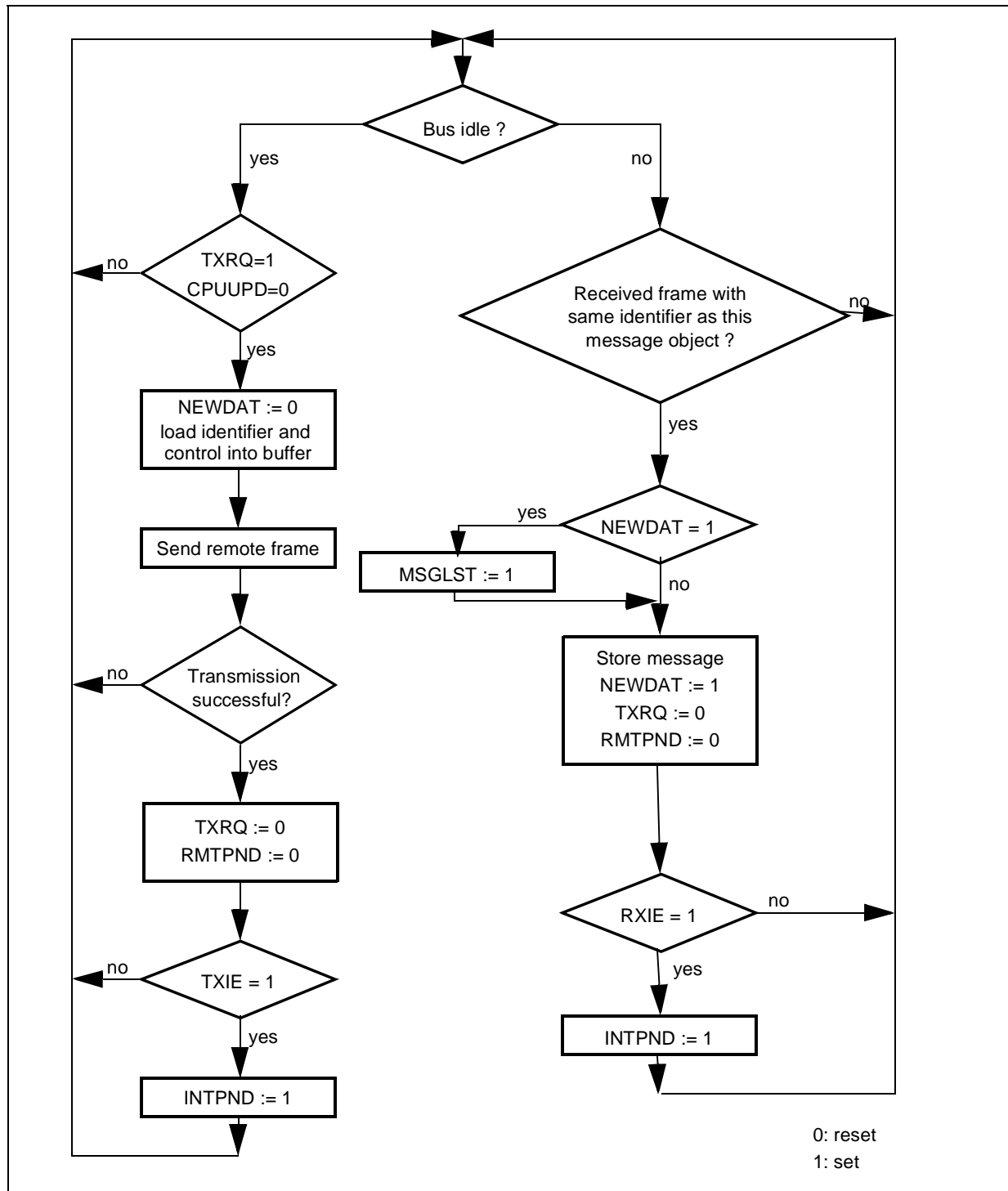


Figure 136 : CPU handling of the last message object

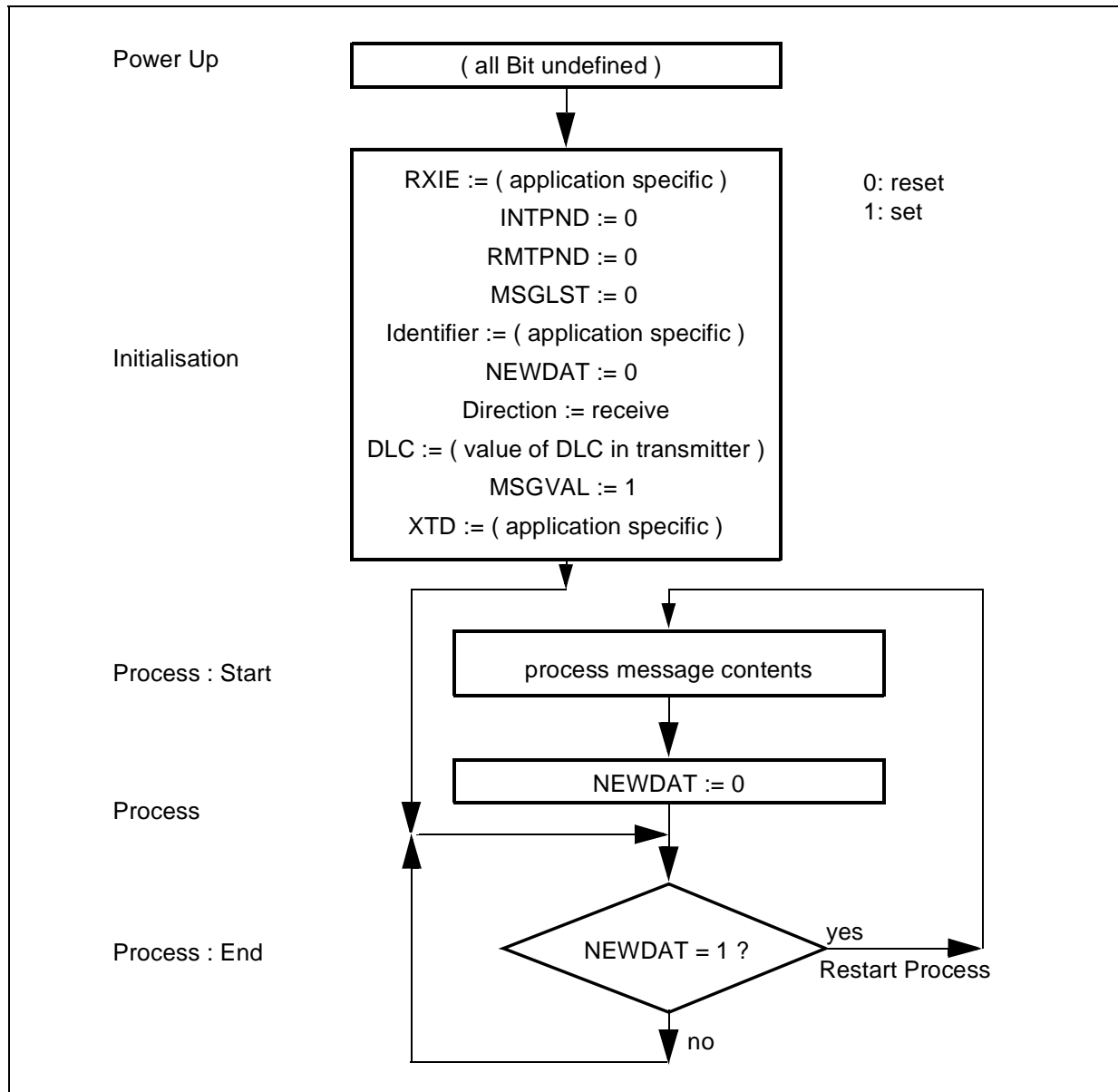


Figure 137 : CPU handling of message objects in receive direction

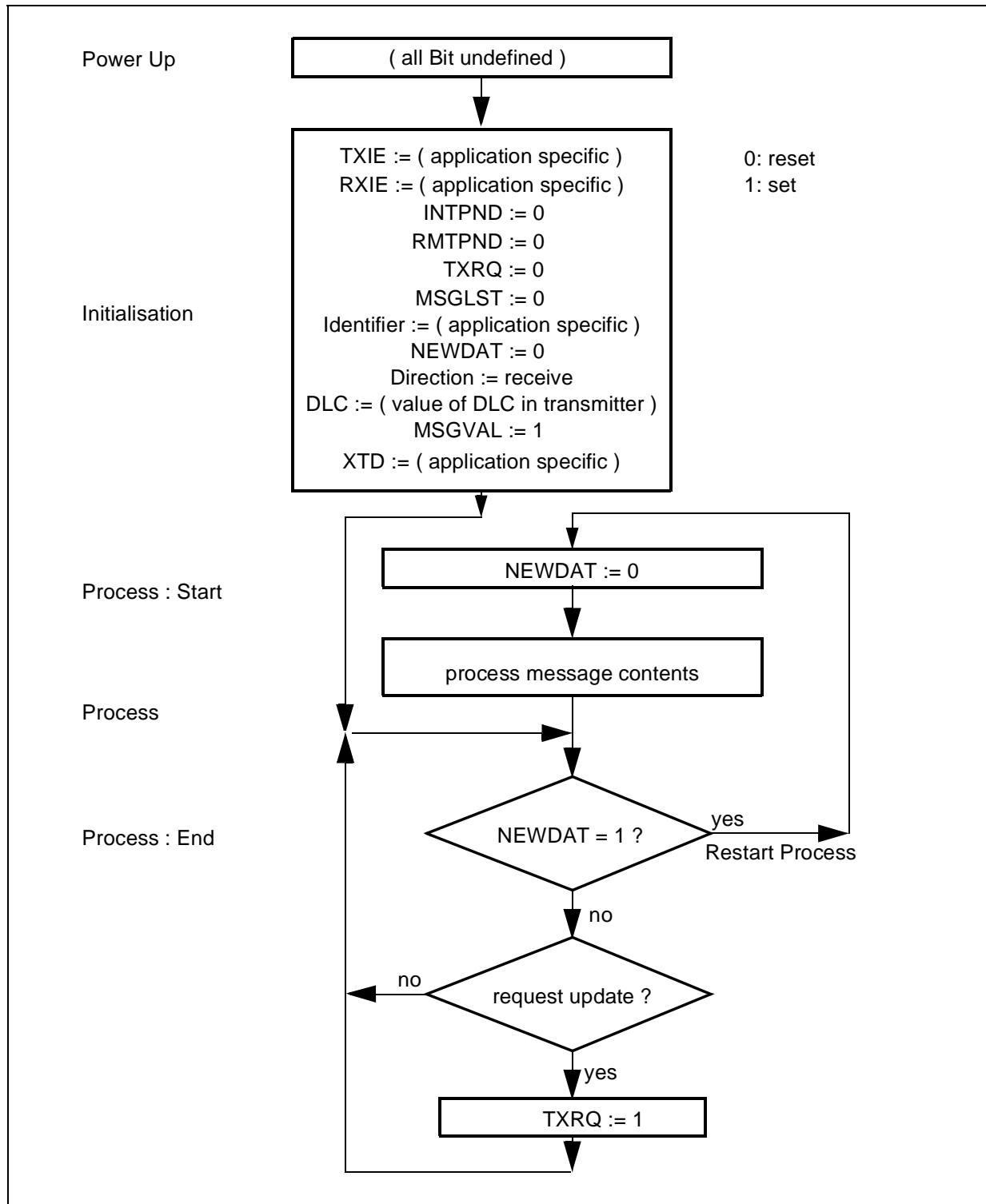


Figure 138 : Handling of the last message object's alternating buffer

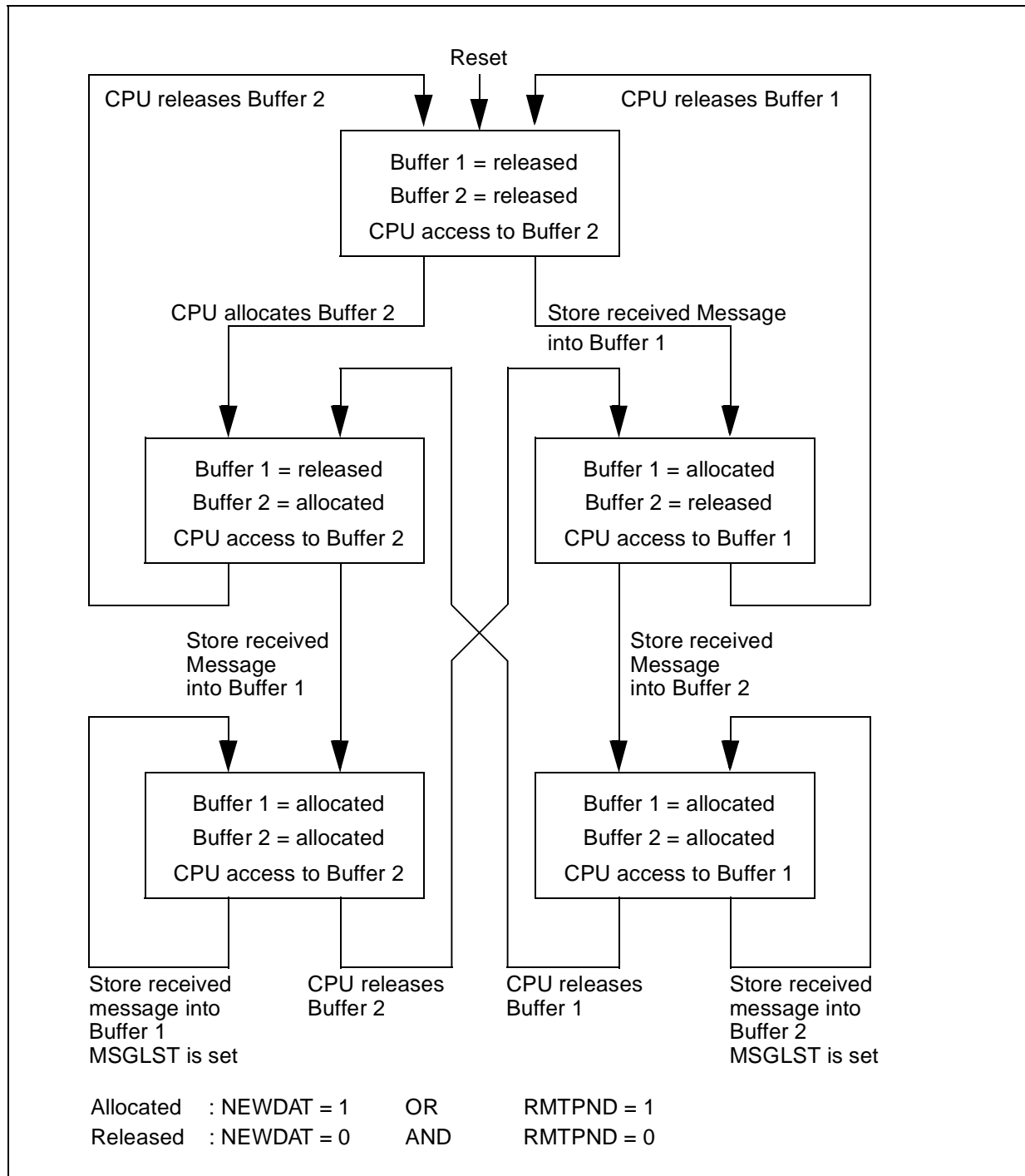


Figure 139 : CAN controller handling of bus recovery sequence

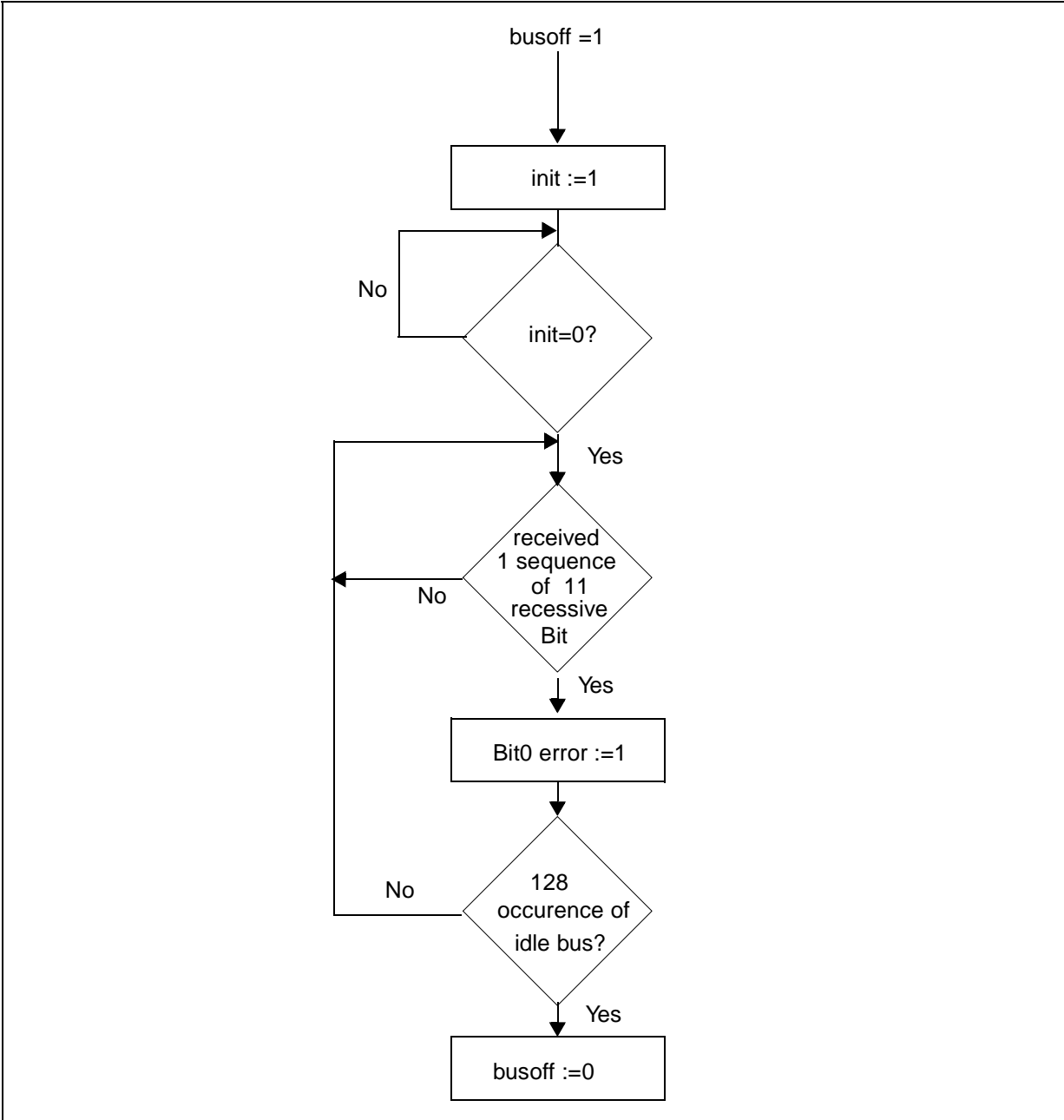
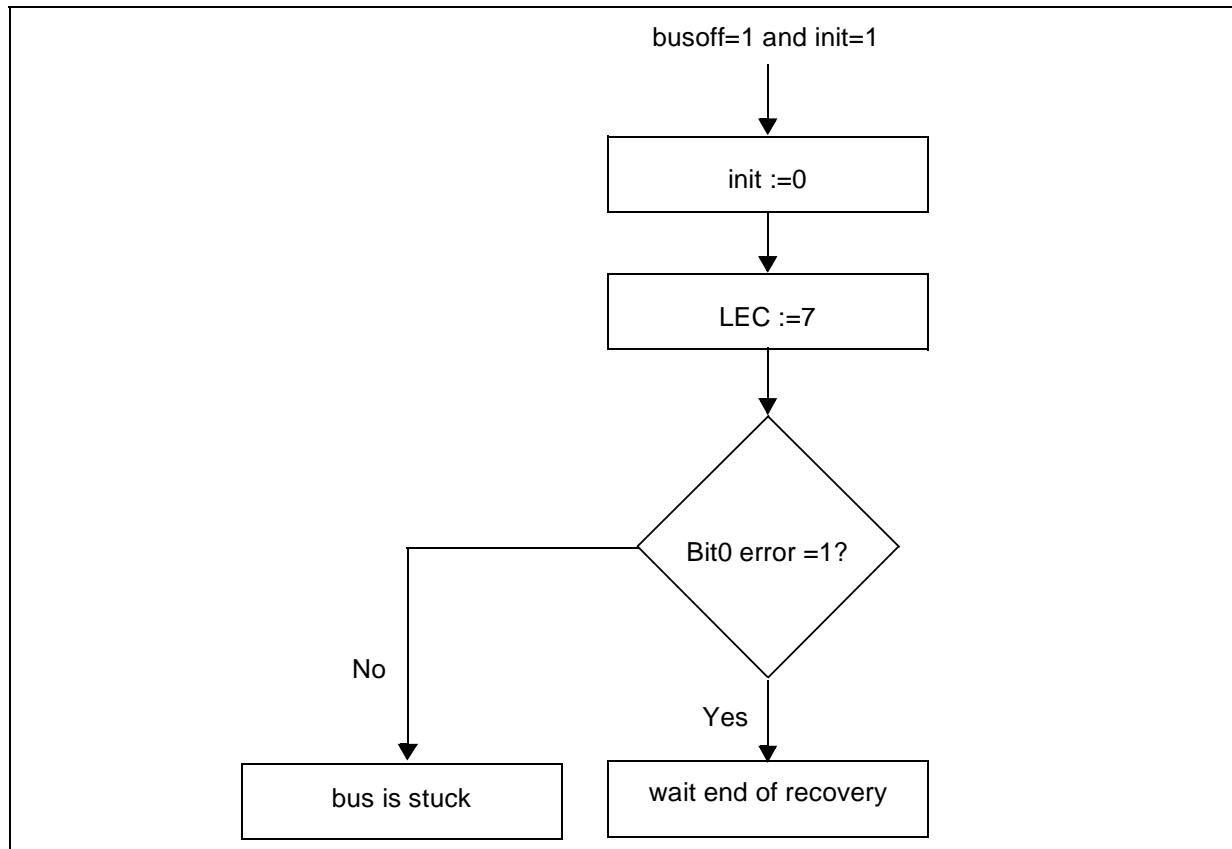


Figure 140 : CPU handling of bus recovery



### 17.6 - Initialization and Reset

The on-chip CAN Module is connected to the XBUS Reset signal XRESET. This signal is activated, when the ST10X167'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 RMTXND 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 Module is implemented as an X-Peripheral and is 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 Module 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 waitstate is used.

The CAN address area starts at 00'EF00h and covers 256 Byte. 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 Module.

Locating the CAN address area to address 00'EF00h in segment 0 has the advantage that the CAN Module is 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 ST10X167 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 Module has to be reconfigured.

**17.7 - CAN Application Interface**

The on-chip CAN Module of the ST10X167 does not incorporate the physical layer connection to the CAN bus. This must be provided externally.

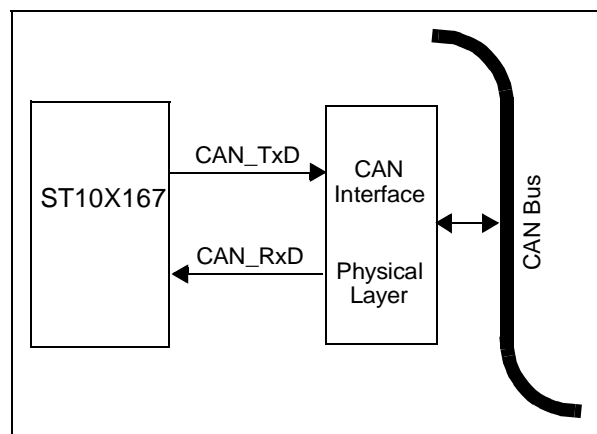
The module's CAN controller is connected to this physical layer (the CAN bus) via two signals:

CAN Signal	Port Pin	Function
CAN_RXD	Port4.5	Receive data from the physical layer of the CAN bus.
CAN_TXD	Port4.6	Transmit data to the physical layer of the CAN 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 segments address lines can be used, reducing the external memory space to 5M Byte.

**Figure 141 : Connection to the CAN bus**







### 18.1 - Types of Reset

Whenever one of the reset conditions occurs, the microcontroller is reset into its predefined default state through an internal reset procedure. When a reset is initiated, pending internal hold states are cancelled. An external bus cycle is aborted, except for a watchdog reset (see description). After that the bus pin drivers and the I/O pin drivers are switched off (tristate).  $\overline{RSTOUT}$  is activated depending on the reset source.

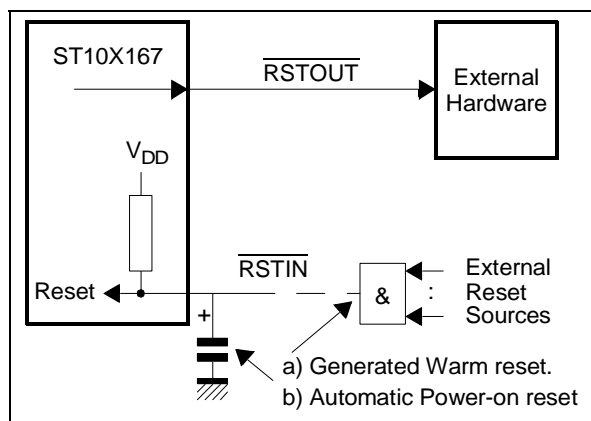
The internal reset procedure takes 516 CPU clock cycles (1032 for asynchronous reset) to perform a complete reset sequence. The reset sequence starts on a watchdog timer overflow, an  $\overline{SRST}$  instruction or when the reset input signal  $\overline{RSTIN}$  is sampled low (hardware reset). The internal reset condition is active :

- Only during the  $\overline{RSTIN}$  input pulse (asynchronous mode), ensure that the duration meets circuit requirements (see Section 18.1.3 - Asynchronous Hardware Reset).
- At least for the duration of the reset sequence and then until the  $\overline{RSTIN}$  input is inactive (synchronous mode).

When this internal reset condition is removed, the reset configuration is latched from PORT0, and pins ALE, RD and WR are driven to their inactive levels.

Bit ADP which selects the Adapt mode is latched with the rising edge of  $\overline{RSTIN}$ .

**Figure 143** : External reset circuitry



#### 18.1.1 - ST10F167 Synchronous Hardware Reset

This synchronous hardware reset is only applicable to ST10F167.

Synchronous hardware reset is triggered when the reset input signal  $\overline{RSTIN}$  is sampled low AND the  $V_{pp}$  pin sampled high. When a synchronous reset is initiated, all pending internal hold states are cancelled and the current internal access

cycle (if any) is completed. Except in the case of a watchdog reset, external bus cycles are aborted. Following this, the internal reset sequence starts, the bus pin drivers and the I/O pin drivers are switched off (tristate), and the PORT0 pins are internally pulled high. The  $\overline{RSTIN}$  pin is driven low for the duration of the reset sequence, which is 516 CPU clock cycles.

To ensure the recognition of the  $\overline{RSTIN}$  signal (latching), it must be held low for at least 2 CPU clock cycles. Also shorter  $\overline{RSTIN}$  pulses may trigger a hardware reset, if they coincide with the latch's sample point. However, it is recommended to keep  $\overline{RSTIN}$  low for at least 1 ms. After the reset sequence has been completed, the  $\overline{RSTIN}$  input is sampled. When the reset input signal is active at that time the internal reset condition is extended until  $\overline{RSTIN}$  gets inactive.

During a synchronous hardware reset the PORT0 inputs for the reset configuration need some time to settle on the required levels, especially if the hardware reset aborts a read operation from an external peripheral. During this settling time the configuration may intermittently be wrong. In such a case also the PLL clock selection may be wrong. It is therefore strongly recommended to provide an external reset pulse of at least 1 ms in order to allow the PLL to settle on the desired CPU clock frequency.

The input  $\overline{RSTIN}$  provides an internal pullup device equalling a resistor of 50 K $\Omega$  to 150 K $\Omega$  (the minimum reset time must be determined by the lowest value). Simply connecting an external capacitor is sufficient for an automatic power-on reset (see b) in Figure 143).  $\overline{RSTIN}$  may also be connected to the output of other logic gates (see a) in Figure 143).

Note A power-on reset requires an active time of two reset sequences (1036 CPU clock cycles) after a stable clock signal is available (plus, about 10...50 ms to allow the on-chip oscillator to stabilize).

#### 18.1.2 - ST10C167 - ST10R167 Synchronous Hardware Reset

This synchronous hardware reset is only applicable to ST10C167 - ST10R167.

Synchronous hardware reset is triggered when the reset input signal  $\overline{RSTIN}$  is sampled low AND the  $V_{pp}$  pin sampled high. To ensure the recognition of the  $\overline{RSTIN}$  signal (latching), it must be held low for at least 2 CPU clock cycles. Also shorter  $\overline{RSTIN}$  pulses may trigger a hardware reset, if they coincide with the latch's sample point.

When a synchronous reset is initiated, all pending internal hold states are cancelled and the current internal access cycle (if any) is completed. Except in the case of a watchdog reset, external bus cycles are aborted. Following this, the internal reset sequence starts, the bus pin drivers and the I/O pin drivers are switched off (tristate), and the PORT0 pins are internally pulled high.

After such a reset, the configuration on Port0 pins is relatched. The configuration needs some time to settle to the required levels, especially if the hardware reset aborts a read operation from an external peripheral. During this settling time the configuration may intermittently be wrong. In such a case also the PLL clock selection may be wrong. **It is therefore strongly recommended to provide an external reset pulse of at least 1 ms in order to allow the PLL to settle on the desired CPU clock frequency.**

The input  $\overline{\text{RSTIN}}$  provides an internal pullup device equalling a resistor of 50 K $\Omega$  to 150 K $\Omega$  (the minimum reset time must be determined by the lowest value). Simply connecting an external capacitor is sufficient for an automatic power-on reset (see b) in Figure 143).  $\overline{\text{RSTIN}}$  may also be connected to the output of other logic gates (see a) in Figure 143).

### 18.1.3 - Asynchronous Hardware Reset

Note This feature does not exist for the ST10F167 device. Asynchronous hardware reset must be used for power on reset of ST10C167 and ST10R167.

Asynchronous reset is invoked by asserting  $\overline{\text{RSTIN}}$  and forcing Vpp low.

While the  $\overline{\text{RSTIN}}$  pin is asserted, a weak internal pull-down is turned on the Vpp pin. When an asynchronous reset is initiated, the microcontroller is immediately (asynchronously) reset into its predefined default state and therefore does not require a stabilized clock signal on XTAL1 pin. When this asynchronous reset condition is removed, the microcontroller starts program execution from memory location 00'0000h in code segment zero.  $\overline{\text{RSTIN}}$  pin must be held low for the whole duration of the circuit internal reset sequence, once the input clock is stabilised and once the PLL synchronised. For application using "exit from power down by external interrupt", this mode is detected by the ST10 during power-up. Constraints on reset duration on the  $\overline{\text{RSTIN}}$  pin are the same as for synchronous reset (1ms for PLL; 10 to 50 ms for on-chip oscillator).



### 18.1.4 - Software Reset

The reset sequence can be triggered at any time by the protected instruction SRST (Software Reset). This instruction can be executed deliberately within a program, e.g. to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure.

Note A software reset disregards the configuration of POL.5...POL.0 and sets Bit "BSL" inactive.

### 18.1.5 - Watchdog Timer Reset

When the watchdog timer is not disabled during the initialization or serviced regularly during program execution it will overflow and trigger the reset sequence. Other than hardware and software reset, the watchdog reset completes a running external bus cycle if this bus cycle either does not use READY at all, or if READY is sampled active (low) after the programmed waitstates. When READY is sampled inactive (high) after the programmed waitstates, the running external bus cycle is aborted. Then the internal reset sequence is started.

Note A watchdog reset disregards the configuration of POL.5...POL.0 and sets Bit "BSL" inactive". The watchdog reset cannot occur while the ST10X167 is in bootstrap loader mode!

### 18.1.6 - Bi-Directional Reset

Note This feature does not exist for the ST10F167 device.

Bi-directional reset converts SW or WDT resets to hardware reset:

- Circuit behaviour is the same as for hardware reset (reset sequence, system start-up configuration from POH and POL)
- Reset sequence is visible at the  $\overline{\text{RSTIN}}$  pin.

Bi-directional reset is disabled during and after hardware reset and is enabled by setting BDRSTEN Bit 3 of the SYSCON register. In bi-directional reset mode, the  $\overline{\text{RSTIN}}$  pin is pulled low for the duration of the internal reset sequence.

Bidirectional reset activates the  $\overline{\text{RSTIN}}$  pin for the duration of an internal reset sequences caused by a WDT reset or a SW Reset, i.e. bidirectional reset transforms an internal WDT reset or SW reset into an external hardware reset with a minimum duration of 516 CPU cycles (if Vpp is high). Consequently, during a WDT reset or SW reset the device behaves as if it was in external hardware reset. Note, the state of the Vpp pin will determine whether the reset is asynchronous or synchronous.

The hardware implementation is shown in Figure 144. The PORT0 sample timing for bidirectional reset is shown in Figure 145.

NOTES:

1. After the execution of the EINIT instruction the bidirectional reset configuration can not be changed.
2. WDTCON Bit 1 of the WDTR register is cleared after a hardware reset.
3. The PORT0 configuration is transparent and latched as for a power-on or PDW Reset.
4. The bootstrap loader can be started by a WDT reset or SW Reset if the bidirectional reset is enabled and P0L.4 is low.
5. If bidirectional reset is enabled then the  $\overline{\text{RSTIN}}$  pin may only be connected to external reset devices with an open drain output driver. A

connection to a push-pull output driver can damage the  $\overline{\text{RSTIN}}$  input.

18.2 - Pins After Reset

After the reset sequence the different groups of pins of the ST10X167 are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can take place or the external control signals are inactive.

The general purpose I/O pins remain in input mode (high impedance) until reprogrammed via software (see Figure 146). The RSTOUT pin remains active (low) until the end of the initialization routine (see description).

Figure 144 : Bi-directional reset hardware implementation

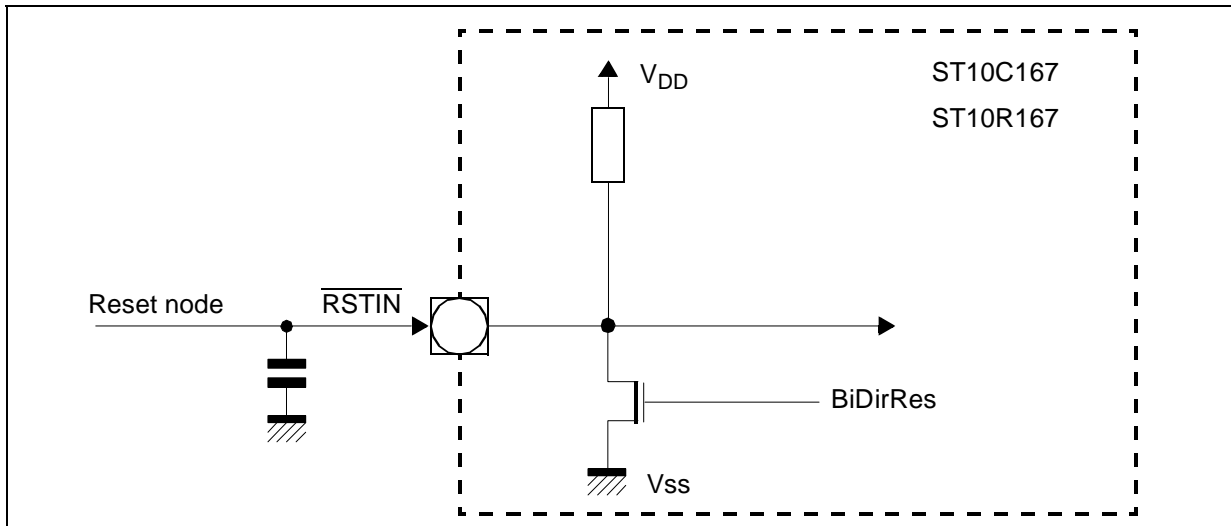


Figure 145 : PORT0 sample timing

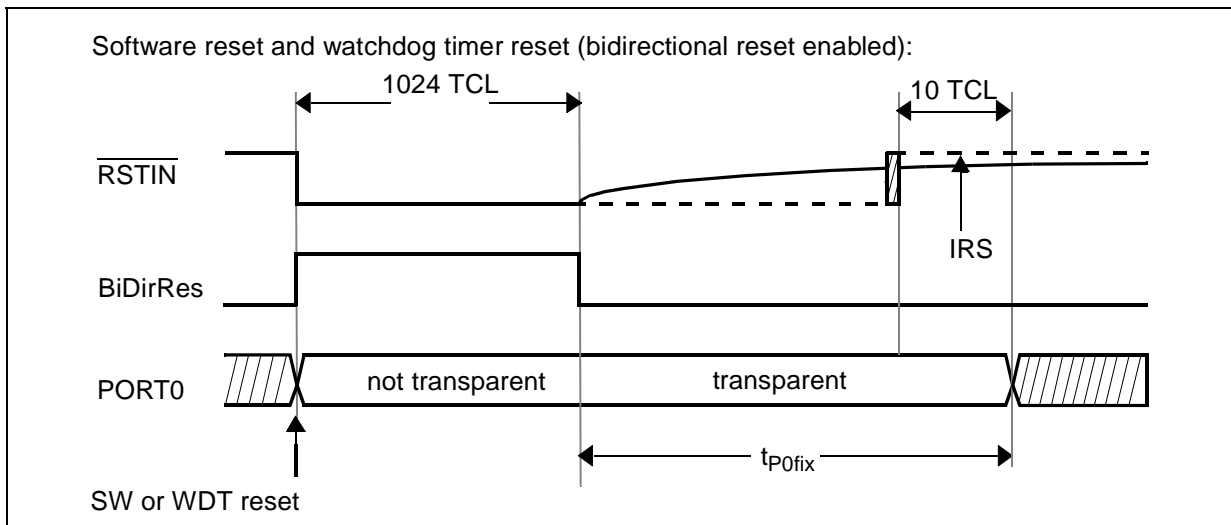
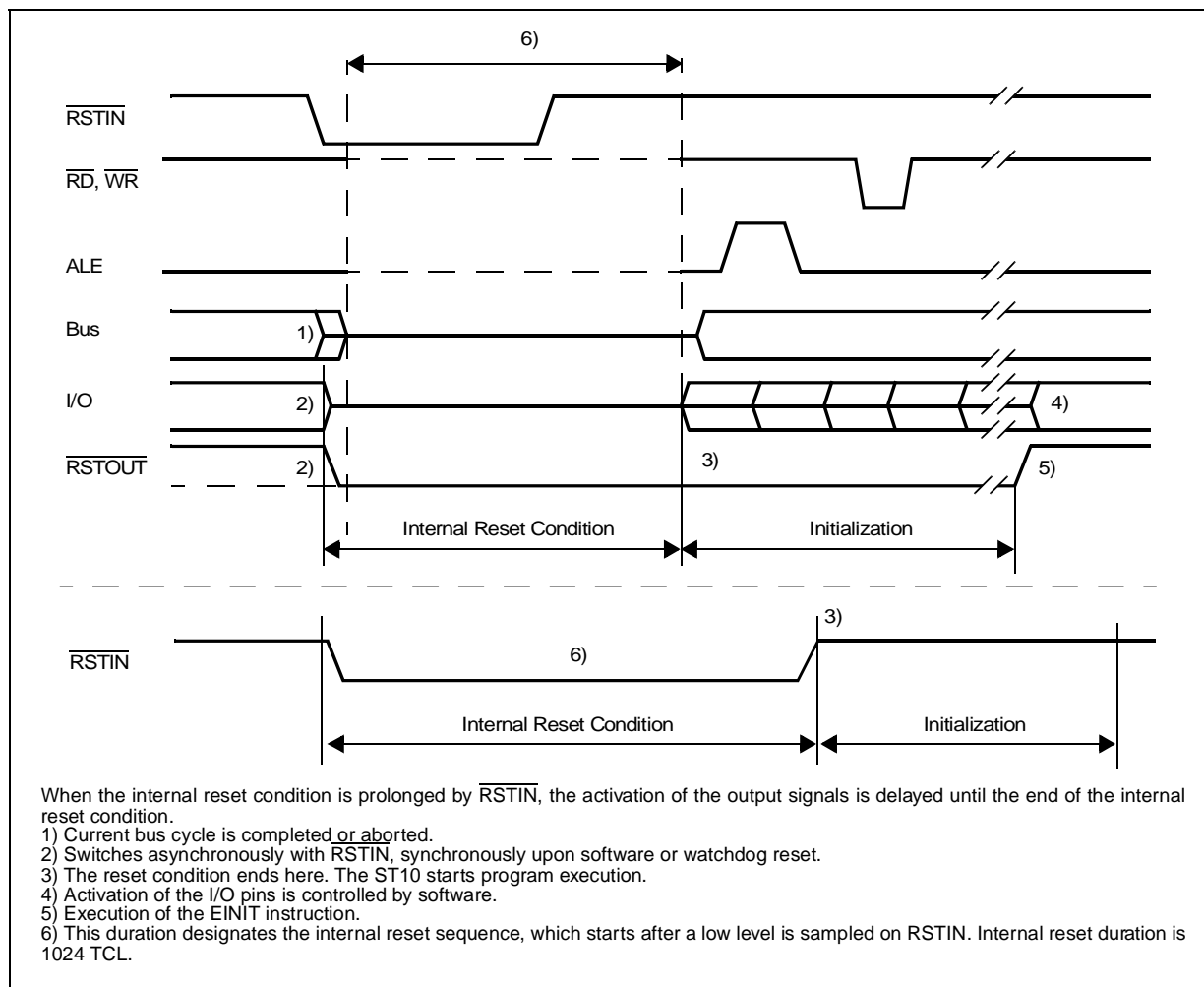


Figure 146 : Synchronous reset



### Reset output pin

The  $\overline{RSTOUT}$  pin generates a reset signal for the system components besides the controller itself.  $\overline{RSTOUT}$  is driven active (low) at the beginning of any reset sequence (triggered by hardware, the  $\overline{SRST}$  instruction or a watchdog timer overflow).  $\overline{RSTOUT}$  stays active (low) beyond the end of the internal reset sequence until the protected EINIT (End of Initialization) instruction is executed (see Figure 146). This allows the complete configuration of the controller including its on-chip peripheral units before releasing the reset signal for the external peripherals of the system.

Note  $\overline{RSTOUT}$  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 WDTCOIN 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 ST10X167 registers**

During the reset sequence the registers of the ST10X167 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 ST10X167'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 ST10X167 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 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 38 and Table 39.

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 Bit 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 CS lines ( $\overline{CS0}$  will be ‘0’, while the other active CS lines will be ‘1’). When no memory accesses above 64 K are to be performed, segmentation may be disabled.

When the on-chip bootstrap loader 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 ST10X167 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 ST10X167 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 ROM. Otherwise it is fetched from external memory. When internal ROM access is enabled after reset in single chip mode (Bit ROMEN='1' in register SYSCON), the software initialization routine may enable and configure the external bus interface before the execution of the EINIT instruction. When external access is enabled after reset, it may be desirable to reconfigure the external bus characteristics, because the SYSCON register is initialized during reset to the slowest possible memory configuration.

To decrease the number of instructions required to initialize the ST10X167, 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 ( $\overline{\text{NMI}}$ ) may occur, even though the interrupt system is still disabled.

In addition, the stack overflow (STKOV) and the stack underflow (STKUN) registers should be initialized. After reset, the CP, SP, and STKUN registers all contain the same reset value 00'FC00h, while the STKOV register contains 00'FA00h. With the default reset initialization, 256 Words of system stack are available, where the system stack selected by the SP grows

downwards from 00'FBFEh, while the register bank selected by the CP grows upwards from 00'FC00h.

Based on the application, the user may wish to initialize portions of the internal memory before normal program operation. Once the register bank has been selected by programming the CP register, the desired portions of the internal memory can easily be initialized via indirect addressing.

At the end of the initialization, the interrupt system may be globally enabled by setting Bit IEN in register PSW. Care must be taken not to enable the interrupt system before the initialization is complete.

The software initialization routine should be terminated with the EINIT instruction. This instruction has been implemented as a protected instruction. Execution of the EINIT instruction disables the action of the DISWDT instruction, disables write accesses to register SYSCON (see note) and causes the  $\overline{\text{RSTOUT}}$  pin to go high. This signal can be used to indicate the end of the initialization routine and the proper operation of the microcontroller to external hardware.

**Note** All configurations regarding register SYSCON (enable CLKOUT, stacksize, etc.) must be selected before the execution of EINIT.

### 18.2.1 - 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  $\overline{\text{EA}}$ ).

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 ST10X167.

Not all Port0 Bit are latched after the end of an internal reset. Depending on the reset type, different Bit 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 144, 145 and 146). To avoid unexpected behavior, the level of the PORT0 configuration input pins should not change while PORT0 is transparent.

**Table 38** : 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
	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 39** : Port0 Bit latched into the different registers after hardware 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
SYSCON	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	WRCFG	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>	X <sup>1</sup>
RPOH	-	-	-	-	-	-	-	-	CLKCFG	CLKCFG	CLKCFG	SALSEL	SALSEL	CSSEL	CSSEL	WRC
BUSCON0						BUSACT0 <sup>2</sup>	ALECTL0 <sup>2</sup>		BUSTYP	BUSTYP						
Internal Logic	To Clock Generator		To Port4 Logic		To Port6 Logic		-	-	-	-	Internal	-	-	Internal	Internal	

- Notes 1. Not latched from Port0.  
2. Bit set if  $\overline{\text{EA}}$  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
-	-	-	-	-	-	-	-		<b>CLKCFG</b>		<b>SALSEL</b>		<b>CSSEL</b>		<b>WRC</b>
									R		R		R		R

Bit	Function																																				
<b>WRC</b>	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																																				
<b>CSSEL</b>	<b>Chip Select Line Selection</b> (Number of active $\overline{CS}$ outputs) 0 0: 3 $\overline{CS}$ lines: $\overline{CS2}...$ $\overline{CS0}$ 0 1: 2 $\overline{CS}$ lines: $\overline{CS1}...$ $\overline{CS0}$ 1 0: No $\overline{CS}$ lines at all 1 1: 5 $\overline{CS}$ lines: $\overline{CS4}...$ $\overline{CS0}$ (Default without pull-downs)																																				
<b>SALSEL</b>	<b>Segment Address Line Selection</b> (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)																																				
<b>CLKCFG</b>	<table border="1"> <thead> <tr> <th>POH.7-5</th><th>ST10C167 - ST10R167</th><th>ST10F167</th><th>Notes</th></tr> </thead> <tbody> <tr> <td></td><td><math>f_{CPU} = f_{XTAL} \times F</math></td><td><math>f_{CPU} = f_{XTAL} \times F</math></td><td></td></tr> <tr> <td>1 1 1</td><td><math>f_{XTAL} \times 4</math></td><td><math>f_{XTAL} \times 4</math></td><td rowspan="4">Default configuration <sup>1</sup></td></tr> <tr> <td>1 1 0</td><td><math>f_{XTAL} \times 3</math></td><td><math>f_{XTAL} \times 4</math></td></tr> <tr> <td>1 0 1</td><td><math>f_{XTAL} \times 2</math></td><td><math>f_{XTAL} \times 4</math></td></tr> <tr> <td>1 0 0</td><td><math>f_{XTAL} \times 5</math></td><td><math>f_{XTAL} \times 4</math></td></tr> <tr> <td>0 1 1</td><td><math>f_{XTAL}</math></td><td><math>f_{XTAL} \times 1</math></td><td>Direct drive</td></tr> <tr> <td>0 1 0</td><td><math>f_{XTAL} \times 1.5</math></td><td><math>f_{XTAL} \times 1</math></td><td></td></tr> <tr> <td>0 0 1</td><td><math>f_{XTAL} \times 0.5</math></td><td><math>f_{XTAL} \times 1</math></td><td rowspan="2">CPU clock via prescaler <sup>2</sup></td></tr> <tr> <td>0 0 0</td><td><math>f_{XTAL} \times 2.5</math></td><td><math>f_{XTAL} \times 1</math></td></tr> </tbody> </table>	POH.7-5	ST10C167 - ST10R167	ST10F167	Notes		$f_{CPU} = f_{XTAL} \times F$	$f_{CPU} = f_{XTAL} \times F$		1 1 1	$f_{XTAL} \times 4$	$f_{XTAL} \times 4$	Default configuration <sup>1</sup>	1 1 0	$f_{XTAL} \times 3$	$f_{XTAL} \times 4$	1 0 1	$f_{XTAL} \times 2$	$f_{XTAL} \times 4$	1 0 0	$f_{XTAL} \times 5$	$f_{XTAL} \times 4$	0 1 1	$f_{XTAL}$	$f_{XTAL} \times 1$	Direct drive	0 1 0	$f_{XTAL} \times 1.5$	$f_{XTAL} \times 1$		0 0 1	$f_{XTAL} \times 0.5$	$f_{XTAL} \times 1$	CPU clock via prescaler <sup>2</sup>	0 0 0	$f_{XTAL} \times 2.5$	$f_{XTAL} \times 1$
POH.7-5	ST10C167 - ST10R167	ST10F167	Notes																																		
	$f_{CPU} = f_{XTAL} \times F$	$f_{CPU} = f_{XTAL} \times F$																																			
1 1 1	$f_{XTAL} \times 4$	$f_{XTAL} \times 4$	Default configuration <sup>1</sup>																																		
1 1 0	$f_{XTAL} \times 3$	$f_{XTAL} \times 4$																																			
1 0 1	$f_{XTAL} \times 2$	$f_{XTAL} \times 4$																																			
1 0 0	$f_{XTAL} \times 5$	$f_{XTAL} \times 4$																																			
0 1 1	$f_{XTAL}$	$f_{XTAL} \times 1$	Direct drive																																		
0 1 0	$f_{XTAL} \times 1.5$	$f_{XTAL} \times 1$																																			
0 0 1	$f_{XTAL} \times 0.5$	$f_{XTAL} \times 1$	CPU clock via prescaler <sup>2</sup>																																		
0 0 0	$f_{XTAL} \times 2.5$	$f_{XTAL} \times 1$																																			

Notes 1. The Bidirectional Reset functionality has no impact on the System Startup Configuration latching.

If the PLL factor or the input clock frequency is changed when PORT0 is transparent, then the PLL needs a PLL synchronization lock time (typical value is 500  $\mu$ s).

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

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 ST10X167 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 39, must remain high during reset in order to ensure proper operation of the ST10X167. 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 ST10X167. In addition also the  $\overline{\text{RSTOUT}}$  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 ST10X167 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 ST10X167 goes into a passive state, which is similar to its state during reset.

The pins of the ST10X167 float to tristate or are deactivated via internal pull-up/pull-down devices, as described for the reset state. In addition also the  $\overline{\text{RSTOUT}}$  pin floats to tristate rather than be driven low, and the on-chip oscillator is switched off.

This mode allows switching a ST10X167 that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original ST10X167 remains in its place. The original ST10X167 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 ST10X167 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 C5h.

**Default:** The ST10X167 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  $\overline{\text{EA}}$ . This allows the configuration of the external bus interface of the ST10X167 even for the first code fetch after reset. The two Bit are copied into Bit field BTYP of register BUSCON0. P0L.7 controls the data bus width, while P0L.6 controls the address output (multiplexed or demultiplexed). This Bit field may be changed via software after reset, if required.

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

PORT0 and PORT1 are automatically switched to the selected bus mode. In multiplexed bus modes PORT0 drives both the 16 Bit intra-segment address and the output data, while PORT1 remains in high impedance state as long as no demultiplexed bus is selected via one of the BUSCON registers. In demultiplexed bus modes PORT1 drives the 16 Bit intra-segment address, while PORT0 or P0L (according to the selected data bus width) drives the output data.

For a 16 Bit data bus  $\overline{\text{BHE}}$  is automatically enabled, for an 8 Bit data bus  $\overline{\text{BHE}}$  is disabled via Bit BYTDIS in register SYSCON.

**Default:** 16 Bit data bus with multiplexed addresses.

Note If an internal start is selected via pin  $\overline{\text{EA}}$ , these two pins are disregarded and Bit field BTYP of register BUSCON0 is cleared.

**Write configuration**

Pin P0H.0 (WRC) selects the initial operation of the control pins  $\overline{\text{WR}}$  and  $\overline{\text{BHE}}$  during reset. When high, this pin selects the standard function, which is  $\overline{\text{WR}}$  control and  $\overline{\text{BHE}}$ . When low, it selects the alternate configuration,  $\overline{\text{WRH}}$  and  $\overline{\text{WRL}}$ . Thus even the first access after a reset can go to a memory controlled via  $\overline{\text{WRH}}$  and  $\overline{\text{WRL}}$ . This Bit is latched in register RP0H and its inverted value is copied into Bit WRCFG in register SYSCON.

**Default:** Standard function ( $\overline{\text{WR}}$  control and  $\overline{\text{BHE}}$ ).

### Chip select lines

Pins P0H.2 and P0H.1 (CSSEL) define the number of active chip select signals during reset.

This allows to select which pins of Port6 drive external  $\overline{CS}$  signals and which are used for general purpose I/O. The two Bit are latched in register RP0H.

**Default:** All 5 chip select lines active ( $\overline{CS4} \dots \overline{CS0}$ ).

CSSEL	Chip Select Lines	Note
1 1	Five: $\overline{CS4} \dots \overline{CS0}$	Default without pull-downs
1 0	None	Port6 pins free for I/O
0 1	Two: $\overline{CS1} \dots \overline{CS0}$	
0 0	Three: $\overline{CS2} \dots \overline{CS0}$	

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 Bit 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	256K Byte (Default without pull-downs)
1 0	Eight: A23...A16	16M Byte (Maximum)
0 1	None	64K Byte (Minimum)
0 0	Four: A19...A16	1M Byte

Even if not all segment address lines are enabled on Port4, the ST10X167 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 256K Byte.

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 Bit are latched in register RP0H.

P0.15-13 (P0H.7-5)	ST10C167 ST10R167 $f_{CPU} = f_{XTAL} \times F$	ST10F167 $f_{CPU} = f_{XTAL} \times F$	Notes
000:	$2.5 \times f_{OSC}$	$1 \times f_{OSC}$	
001:	$0.5 \times f_{OSC}$	$1 \times f_{OSC}$	Prescaler (not for ST10F167)
010:	$1.5 \times f_{OSC}$	$1 \times f_{OSC}$	
011:	$1 \times f_{OSC}$	$1 \times f_{OSC}$	Direct drive
100:	$5 \times f_{OSC}$	$4 \times f_{OSC}$	
101:	$2 \times f_{OSC}$	$4 \times f_{OSC}$	
110:	$3 \times f_{OSC}$	$4 \times f_{OSC}$	
111:	$4 \times f_{OSC}$	$4 \times f_{OSC}$	Default configuration

- Notes
1. The external clock input range refers to a CPU clock range of 10 to 25 MHz.
  2. The maximum depends on the duty cycle of the external clock signal.
  3. **Default:** On-chip PLL is active with a factor of 1:4.
  4. Watch the different requirements for frequency and duty cycle of the oscillator input clock for the possible selections.

## 19 - POWER REDUCTION MODES

Two different power reduction modes have been implemented in the ST10X167.

- **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.

### 19.1 - Idle Mode

The power consumption of the ST10X167 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

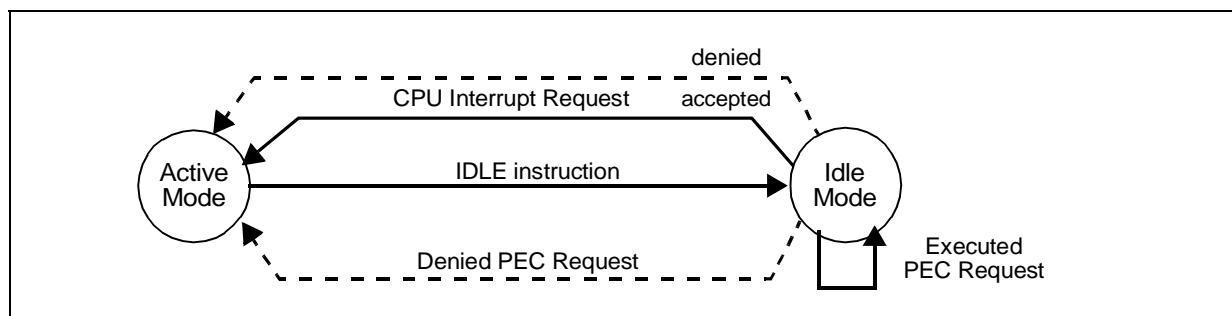
interrupt service routine is executed, the CPU continues executing the program with the instruction following the IDLE instruction. Otherwise, if the interrupt request cannot be serviced because of a too low priority or a globally disabled interrupt system, the CPU immediately resumes normal program 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 147).

Idle mode can also be terminated by a Non-Maskable Interrupt, with a high to low transition on the 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 147 :** 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.

## 19.2 - 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 (ST10F167 can only use protected power down mode)
- Interruptible Power Down Mode (not available for ST10F167).

The power down mode is selected by Bit 5 - PWDCFG in SYSCON register.

SYSCON (FF12h / 89h)										SFR					Reset Value: 0X00h <sup>1</sup>				
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	RW				

Note 1. Reset value is 0XX0h for ST10F167.

Bit	Function
PWDCFG not allocated in ST10F167	<b>Power Down Mode Configuration Control</b> '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 occur 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.

### 19.2.1 - Protected Power Down Mode

This mode is selected by setting Bit PWDCFG in the SYSCON register to '0' (not needed for ST10F167).

**Entering power down mode** can only be achieved if the 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 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 NMI pin is still low at this time, Power Down mode will be entered, otherwise program execution continues. During

power down, the voltage at the  $V_{DD}$  pins can be lowered to 2.5 V while preserving the contents of the internal RAM.

**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.

### 19.3 - Interruptible Power Down Mode

This mode is selected by setting the Bit PWDCFG in register SYSCON to '1'. This mode is not available for ST10F167.

**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)	<p><b>External Interrupt x Edge Selection Field (x=7...0)</b></p> <p>0 0: Fast external interrupts disabled: standard mode EXxIN pin not taken in account for entering/exiting Power Down mode.</p> <p>0 1: Interrupt on positive edge (rising) Enter Power Down mode if EXxIN = '0', exit if EXxIN = '1' (ref as 'high' active level)</p> <p>1 0: Interrupt on negative edge (falling) Enter Power Down mode if EXxIN = '1', exit if EXxIN = '0' (ref as 'low' active level)</p> <p>1 1: Interrupt on any edge (rising or falling) Always enter Power Down mode, exit if EXxIN level changed.</p>

**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 RSTIN, or one of the enabled EXxIN pins (Fast External Interrupt).

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 148.

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 149).

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 Vpp 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 148** : Delay with RC on Vpp pin

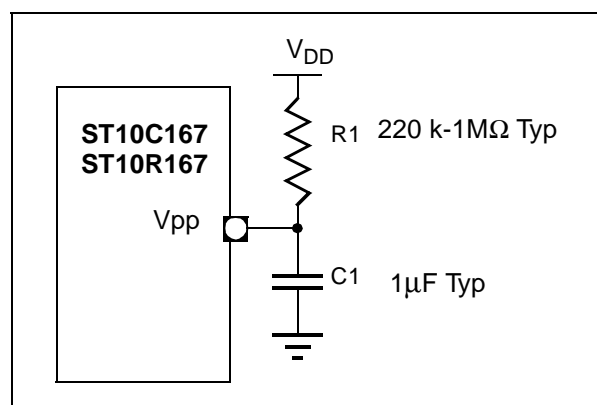


Figure 149 : Simplified powerdown exit circuitry

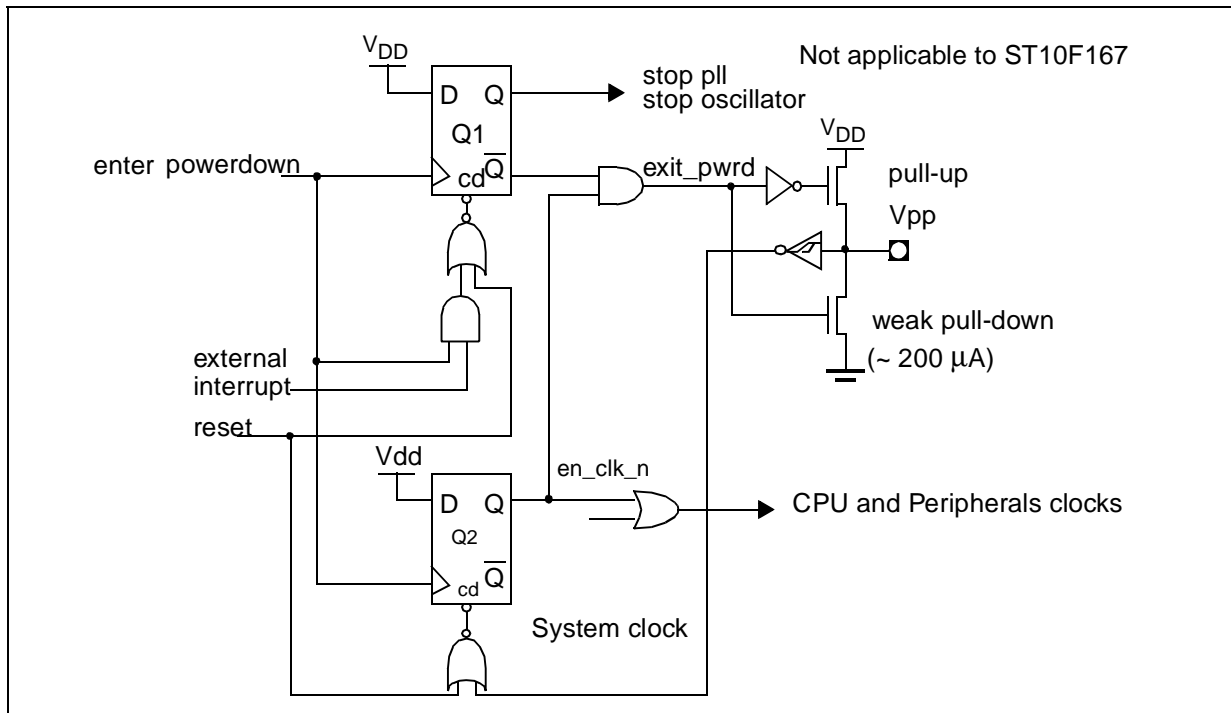
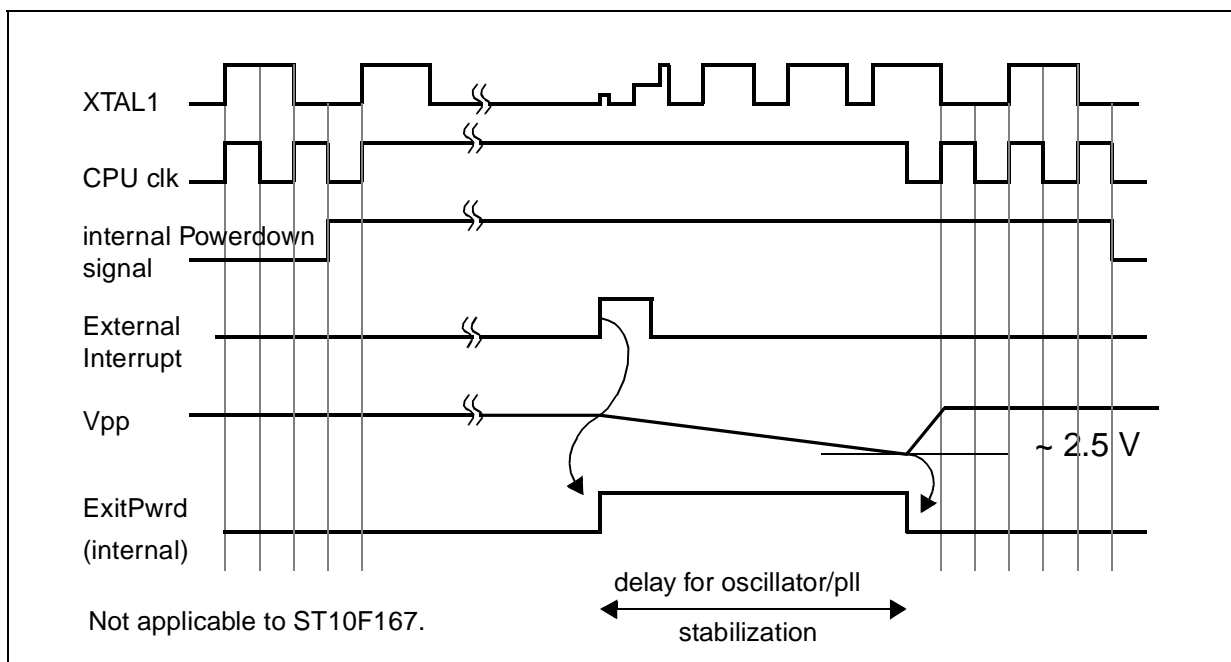


Figure 150 : Powerdown exit sequence when using an external interrupt (PLL x 2)



### 19.4 - 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 (WR), or to a defined state which is based on the last bus access (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 Bit of the last address if a demultiplexed 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 40 summarizes the state of all ST10X167 output pins during Idle and Power Down mode.

**Table 40** : Output pin state during idle and powerdown modes

ST10X167 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 <sup>2</sup> / Float	Port Latch Data	A15...A8 <sup>2</sup> / Float
PORT1	Port Latch Data	Last Address <sup>3</sup> / Port Latch Data	Port Latch Data	Last Address <sup>3</sup> / 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 <sup>4</sup>	Port Latch Data	Last value <sup>4</sup>
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 demultiplexed buses.

4. The CS signal that corresponds to the last address remains active (low), all other enabled CS signals remain inactive (high). By accessing an on-chip X-Peripheral prior to entering a power save mode all external CS signals can be deactivated.



## 20 - REGISTER SET

This section summarizes all registers implemented in the ST10X167, 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.

### 20.1 - 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	<b>hw bit</b>	read only	std bit	<b>hw bit</b>	bitfield			bitfield			
					W	RW	R	RW	RW			RW			RW	

Bit	Function
bit(field) name	<b>Explanation of bit(field) name</b> 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	<b>hw bit</b>	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
	<b>0/1</b> : defined
	<b>X</b> : undefined (undefined ('X') after power up)
	<b>U</b> : unchanged

**hwbit** bit that are set/cleared by hardware are written in **bold**

### 20.2 - 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 41** : 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 42** : 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

### 20.3 - Special Function Registers Ordered by Name

The following table lists all SFRs which are implemented in the ST10X167 in alphabetical order.

Bit-addressable SFRs are marked with the letter

“b” in column “Name”.

SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter “E” in column “Physical Address”. Registers within on-chip X-Peripherals (CAN) are marked with the letter “X” in column “Physical Address”

**Table 43** : Special function registers ordered by name

Name	Physical Address	8 bit Address	Description	Reset Value
ADCIC b	FF98h	CCh	A/D Converter End of Conversion Interrupt Control Register	0000h
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 Reg	0000h
BUSCON0 b	FF0Ch	86h	Bus Configuration Register 0	0XX0h
BUSCON1 b	FF14h	8Ah	Bus Configuration Register 1	0000h
BUSCON2 b	FF16h	8Bh	Bus Configuration Register 2	0000h
BUSCON3 b	FF18h	8Ch	Bus Configuration Register 3	0000h
BUSCON4 b	FF1Ah	8Dh	Bus Configuration Register 4	0000h
CAPREL	FE4Ah	25h	GPT2 Capture/Reload Register	0000h
CC8IC b	FF88h	C4h	EX0IN Interrupt Control Register	0000h
CC0	FE80h	40h	CAPCOM Register 0	0000h
CC0IC b	FF78h	BCh	CAPCOM Register 0 Interrupt Control Register	0000h
CC1	FE82h	41h	CAPCOM Register 1	0000h
CC1IC b	FF7Ah	BDh	CAPCOM Register 1 Interrupt Control Register	0000h
CC2	FE84h	42h	CAPCOM Register 2	0000h
CC2IC b	FF7Ch	BEh	CAPCOM Register 2 Interrupt Control Register	0000h
CC3	FE86h	43h	CAPCOM Register 3	0000h
CC3IC b	FF7Eh	BFh	CAPCOM Register 3 Interrupt Control Register	0000h
CC4	FE88h	44h	CAPCOM Register 4	0000h
CC4IC b	FF80h	C0h	CAPCOM Register 4 Interrupt Control Register	0000h
CC5	FE8Ah	45h	CAPCOM Register 5	0000h
CC5IC b	FF82h	C1h	CAPCOM Register 5 Interrupt Control Register	0000h
CC6	FE8Ch	46h	CAPCOM Register 6	0000h
CC6IC b	FF84h	C2h	CAPCOM Register 6 Interrupt Control Register	0000h
CC7	FE8Eh	47h	CAPCOM Register 7	0000h
CC7IC b	FF86h	C3h	CAPCOM Register 7 Interrupt Control Register	0000h

Table 43 : Special function registers ordered by name (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
CC8	FE90h	48h	CAPCOM Register 8	0000h
CC8IC b	FF88h	C4h	CAPCOM Register 8 Interrupt Control Register	0000h
CC9	FE92h	49h	CAPCOM Register 9	0000h
CC9IC b	FF8Ah	C5h	CAPCOM Register 9 Interrupt Control Register	0000h
CC10	FE94h	4Ah	CAPCOM Register 10	0000h
CC10IC b	FF8Ch	C6h	CAPCOM Register 10 Interrupt Control Register	0000h
CC11	FE96h	4Bh	CAPCOM Register 11	0000h
CC11IC b	FF8Eh	C7h	CAPCOM Register 11 Interrupt Control Register	0000h
CC12	FE98h	4Ch	CAPCOM Register 12	0000h
CC12IC b	FF90h	C8h	CAPCOM Register 12 Interrupt Control Register	0000h
CC13	FE9Ah	4Dh	CAPCOM Register 13	0000h
CC13IC b	FF92h	C9h	CAPCOM Register 13 Interrupt Control Register	0000h
CC14	FE9Ch	4Eh	CAPCOM Register 14	0000h
CC14IC b	FF94h	CAh	CAPCOM Register 14 Interrupt Control Register	0000h
CC15	FE9Eh	4Fh	CAPCOM Register 15	0000h
CC15IC b	FF96h	CBh	CAPCOM Register 15 Interrupt Control Register	0000h
CC16	FE60h	30h	CAPCOM Register 16	0000h
CC16IC b	F160h E	B0h	CAPCOM Register 16 Interrupt Control Register	0000h
CC17	FE62h	31h	CAPCOM Register 17	0000h
CC17IC b	F162h E	B1h	CAPCOM Register 17 Interrupt Control Register	0000h
CC18	FE64h	32h	CAPCOM Register 18	0000h
CC18IC b	F164h E	B2h	CAPCOM Register 18 Interrupt Control Register	0000h
CC19	FE66h	33h	CAPCOM Register 19	0000h
CC19IC b	F166h E	B3h	CAPCOM Register 19 Interrupt Control Register	0000h
CC20	FE68h	34h	CAPCOM Register 20	0000h
CC20IC b	F168h E	B4h	CAPCOM Register 20 Interrupt Control Register	0000h
CC21	FE6Ah	35h	CAPCOM Register 21	0000h
CC21IC b	F16Ah E	B5h	CAPCOM Register 21 Interrupt Control Register	0000h
CC22	FE6Ch	36h	CAPCOM Register 22	0000h
CC22IC b	F16Ch E	B6h	CAPCOM Register 22 Interrupt Control Register	0000h
CC23	FE6Eh	37h	CAPCOM Register 23	0000h
CC23IC b	F16Eh E	B7h	CAPCOM Register 23 Interrupt Control Register	0000h
CC24	FE70h	38h	CAPCOM Register 24	0000h
CC24IC b	F170h E	B8h	CAPCOM Register 24 Interrupt Control Register	0000h
CC25	FE72h	39h	CAPCOM Register 25	0000h
CC25IC b	F172h E	B9h	CAPCOM Register 25 Interrupt Control Register	0000h
CC26	FE74h	3Ah	CAPCOM Register 26	0000h

Table 43 : Special function registers ordered by name (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
CC26IC b	F174h E	BAh	CAPCOM Register 26 Interrupt Control Register	0000h
CC27	FE76h	3Bh	CAPCOM Register 27	0000h
CC27IC b	F176h E	BBh	CAPCOM Register 27 Interrupt Control Register	0000h
CC28	FE78h	3Ch	CAPCOM Register 28	0000h
CC28IC b	F178h E	BCh	CAPCOM Register 28 Interrupt Control Register	0000h
CC29	FE7Ah	3Dh	CAPCOM Register 29	0000h
CC29IC b	F184h E	C2h	CAPCOM Register 29 Interrupt Control Register	0000h
CC30	FE7Ch	3Eh	CAPCOM Register 30	0000h
CC30IC b	F18Ch E	C6h	CAPCOM Register 30 Interrupt Control Register	0000h
CC31	FE7Eh	3Fh	CAPCOM Register 31	0000h
CC31IC b	F194h E	CAh	CAPCOM Register 31 Interrupt 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
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	0000h
CSP	FE08h	04h	CPU Code Segment Pointer Register (read only)	0000h
DP0L b	F100h E	80h	P0L Direction Control Register	00h
DP0H b	F102h E	81h	P0h Direction Control Register	00h
DP1L b	F104h E	82h	P1L Direction Control Register	00h
DP1H b	F106h E	83h	P1h Direction Control Register	00h
DP2 b	FFC2h	E1h	Port2 Direction Control Register	0000h
DP3 b	FFC6h	E3h	Port3 Direction Control Register	0000h
DP4 b	FFCAh	E5h	Port4 Direction Control Register	00h
DP6 b	FFCEh	E7h	Port6 Direction Control Register	00h
DP7 b	FFD2h	E9h	Port7 Direction Control Register	00h
DP8 b	FFD6h	EBh	Port8 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

Table 43 : Special function registers ordered by name (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
IDCHIP	F07Ch E	3Eh	Device Identifier Register (not in ST10F167)	Ref dsheet
IDMANUF	F07Eh E	3Fh	Manufacturer Identifier Register (not in ST10F167)	0020h
IDMEM	F07Ah E	3Dh	On-chip Memory Identifier Register (not in ST10F167)	Ref dsheet
IDPROG	F078h E	3Ch	Programming Voltage Identifier Register (not in ST10F167)	Ref dsheet
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
ODP2 b	F1C2h E	E1h	Port2 Open Drain Control Register	0000h
ODP3 b	F1C6h E	E3h	Port3 Open Drain Control Register	0000h
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
ONES	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	Port2 Register	0000h
P3 b	FFC4h	E2h	Port3 Register	0000h
P4 b	FFC8h	E4h	Port4 Register (8 bit)	00h
P5 b	FFA2h	D1h	Port5 Register (read only)	XXXXh
P6 b	FFCCh	E6h	Port6 Register (8 bit)	00h
P7 b	FFD0h	E8h	Port7 Register (8 bit)	00h
P8 b	FFD4h	EAh	Port8 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	F1C4h E	E2h	Port Input Threshold Control Register	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

Table 43 : Special function registers ordered by name (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
PSW	b FF10h	88h	CPU Program Status Word	0000h
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	0000h
RP0H	b F108h E	84h	System Start-up Configuration Register (Rd. 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	0000h
S0RBUF	FEB2h	59h	Serial Channel 0 Receive Buffer Register (read only)	XXh
S0RIC	b FF6Eh	B7h	Serial Channel 0 Receive Interrupt Control Register	0000h
S0TBIC	b F19Ch E	CEh	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000h
S0TBUF	FEB0h	58h	Serial Channel 0 Transmit Buffer Reg (write only)	00h
S0TIC	b FF6Ch	B6h	Serial Channel 0 Transmit Interrupt Control Register	0000h
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	0000h
SSCRB	F0B2h E	59h	SSC Receive Buffer (read only)	XXXXh
SSCRIC	b FF74h	BAh	SSC Receive Interrupt Control Register	0000h
SSCTB	F0B0h E	58h	SSC Transmit Buffer (write only)	0000h
SSCTIC	b FF72h	B9h	SSC Transmit Interrupt Control Register	0000h
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
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	0000h
T0REL	FE54h	2Ah	CAPCOM Timer 0 Reload Register	0000h
T1	FE52h	29h	CAPCOM Timer 1 Register	0000h

Table 43 : Special function registers ordered by name (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
T1IC	b FF9Eh	CFh	CAPCOM Timer 1 Interrupt Control Register	0000h
T1REL	FE56h	2Bh	CAPCOM Timer 1 Reload Register	0000h
T2	FE40h	20h	GPT1 Timer 2 Register	0000h
T2CON	b FF40h	A0h	GPT1 Timer 2 Control Register	0000h
T2IC	b FF60h	B0h	GPT1 Timer 2 Interrupt Control Register	0000h
T3	FE42h	21h	GPT1 Timer 3 Register	0000h
T3CON	b FF42h	A1h	GPT1 Timer 3 Control Register	0000h
T3IC	b FF62h	B1h	GPT1 Timer 3 Interrupt Control Register	0000h
T4	FE44h	22h	GPT1 Timer 4 Register	0000h
T4CON	b FF44h	A2h	GPT1 Timer 4 Control Register	0000h
T4IC	b FF64h	B2h	GPT1 Timer 4 Interrupt Control Register	0000h
T5	FE46h	23h	GPT2 Timer 5 Register	0000h
T5CON	b FF46h	A3h	GPT2 Timer 5 Control Register	0000h
T5IC	b FF66h	B3h	GPT2 Timer 5 Interrupt Control Register	0000h
T6	FE48h	24h	GPT2 Timer 6 Register	0000h
T6CON	b FF48h	A4h	GPT2 Timer 6 Control Register	0000h
T6IC	b FF68h	B4h	GPT2 Timer 6 Interrupt Control Register	0000h
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	0000h
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	0000h
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	FFAEh	D7h	Watchdog Timer Control Register	000xh
XP0IC	b F186h E	C3h	CAN Module Interrupt Control Register	0000h
XP1IC	b F18Eh E	C7h	X-Peripheral 1 Interrupt Control Register	0000h
XP2IC	b F196h E	CBh	X-Peripheral 2 Interrupt Control Register	0000h
XP3IC	b F19Eh E	CFh	PLL unlock Interrupt Control Register	0000h
ZEROS	b FF1Ch	8Eh	Constant Value 0's Register (read only)	0000h



## 20.4 - Registers Ordered by Address

The following table lists all SFRs which are implemented in the ST10X167 ordered by their physical address. **Bit-addressable** SFRs are marked with the letter “b” in column “Name”.

SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter “E” in column “Physical Address”. Registers within on-chip X-Peripherals (CAN) are marked with the letter “X” in column “Physical Address”.

**Table 44** : Registers ordered by address

Name	Physical Address	8 bit Address	Description	Reset Value
PT0	F030h E	18h	PWM Module Up/Down Counter 0	0000h
PT1	F032h E	19Hh	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 (not in ST10F167)	Ref dsheet
IDMEM	F07Ah E	3Dh	On-chip Memory Identifier Register (not in ST10F167)	Ref dsheet
IDCHIP	F07Ch E	3Eh	Device Identifier Register (not in ST10F167)	Ref dsheet
IDMANUF	F07Eh E	3Fh	Manufacturer Identifier Register (not in ST10F167)	0020h
ADDAT2	F0A0h E	50h	A/D Converter 2 Result Register	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
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	0000h
CC17IC	b F162h E	B1h	CAPCOM Register 17 Interrupt Control Register	0000h
CC18IC	b F164h E	B2h	CAPCOM Register 18 Interrupt Control Register	0000h
CC19IC	b F166h E	B3h	CAPCOM Register 19 Interrupt Control Register	0000h
CC20IC	b F168h E	B4h	CAPCOM Register 20 Interrupt Control Register	0000h
CC21IC	b F16Ah E	B5h	CAPCOM Register 21 Interrupt Control Register	0000h
CC22IC	b F16Ch E	B6h	CAPCOM Register 22 Interrupt Control Register	0000h

Table 44 : Registers ordered by address (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
CC23IC	b F16Eh E	B7h	CAPCOM Register 23 Interrupt Control Register	0000h
CC24IC	b F170h E	B8h	CAPCOM Register 24 Interrupt Control Register	0000h
CC25IC	b F172h E	B9h	CAPCOM Register 25 Interrupt Control Register	0000h
CC26IC	b F174h E	BAh	CAPCOM Register 26 Interrupt Control Register	0000h
CC27IC	b F176h E	BBh	CAPCOM Register 27 Interrupt Control Register	0000h
CC28IC	b F178h E	BCh	CAPCOM Register 28 Interrupt Control Register	0000h
T7IC	b F17Ah E	BEh	CAPCOM Timer 7 Interrupt Control Register	0000h
T8IC	b F17Ch E	BFh	CAPCOM Timer 8 Interrupt Control Register	0000h
PWMIC	b F17Eh E	BFh	PWM Module Interrupt Control Register	0000h
CC29IC	b F184h E	C2h	CAPCOM Register 29 Interrupt Control Register	0000h
XP0IC	b F186h E	C3h	CAN Module Interrupt Control Register	0000h
CC30IC	b F18Ch E	C6h	CAPCOM Register 30 Interrupt Control Register	0000h
XP1IC	b F18Eh E	C7h	X-Peripheral 1 Interrupt Control Register	0000h
CC31IC	b F194h E	CAh	CAPCOM Register 31 Interrupt Control Register	0000h
XP2IC	b F196h E	CBh	X-Peripheral 2 Interrupt Control Register	0000h
S0TBIC	b F19Ch E	CEh	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000h
XP3IC	b F19Eh E	CFh	PLL unlock Interrupt Control Register	0000h
EXICON	b F1C0h E	E0h	External Interrupt Control Register	0000h
ODP2	b F1C2h E	E1h	Port2 Open Drain Control Register	0000h
PICON	F1C4h E	E2h	Port Input Threshold Control Register	0000h
ODP3	b F1C6h E	E3h	Port3 Open Drain Control Register	0000h
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
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
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

Table 44 : Registers ordered by address (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
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
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
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

Table 44 : Registers ordered by address (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
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)	00h
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
BUSCON0	b FF0Ch	86h	Bus Configuration Register 0	0XX0h
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

**Table 44** : Registers ordered by address (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
ZEROS	b FF1Ch	8Eh	Constant Value 0's Register (read only)	0000h
ONES	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	0000h
T3IC	b FF62h	B1h	GPT1 Timer 3 Interrupt Control Register	0000h
T4IC	b FF64h	B2h	GPT1 Timer 4 Interrupt Control Register	0000h
T5IC	b FF66h	B3h	GPT2 Timer 5 Interrupt Control Register	0000h
T6IC	b FF68h	B4h	GPT2 Timer 6 Interrupt Control Register	0000h
CRIC	b FF6Ah	B5h	GPT2 CAPREL Interrupt Control Register	0000h
S0TIC	b FF6Ch	B6h	Serial Channel 0 Transmit Interrupt Control Register	0000h
S0RIC	b FF6Eh	B7h	Serial Channel 0 Receive Interrupt Control Register	0000h
S0EIC	b FF70h	B8h	Serial Channel 0 Error Interrupt Control Register	0000h
SSCTIC	b FF72h	B9h	SSC Transmit Interrupt Control Register	0000h
SSCRIC	b FF74h	BAh	SSC Receive Interrupt Control Register	0000h
SSCEIC	b FF76h	BBh	SSC Error Interrupt Control Register	0000h
CC0IC	b FF78h	BCh	CAPCOM Register 0 Interrupt Control Register	0000h
CC1IC	b FF7Ah	BDh	CAPCOM Register 1 Interrupt Control Register	0000h
CC2IC	b FF7Ch	BEh	CAPCOM Register 2 Interrupt Control Register	0000h
CC3IC	b FF7Eh	BFh	CAPCOM Register 3 Interrupt Control Register	0000h
CC4IC	b FF80h	C0h	CAPCOM Register 4 Interrupt Control Register	0000h
CC5IC	b FF82h	C1h	CAPCOM Register 5 Interrupt Control Register	0000h

Table 44 : Registers ordered by address (continued)

Name	Physical Address	8 bit Address	Description	Reset Value
CC6IC	b FF84h	C2h	CAPCOM Register 6 Interrupt Control Register	0000h
CC7IC	b FF86h	C3h	CAPCOM Register 7 Interrupt Control Register	0000h
CC8IC	b FF88h	C4h	EX0IN Interrupt Control Register	0000h
CC8IC	b FF88h	C4h	CAPCOM Register 8 Interrupt Control Register	0000h
CC9IC	b FF8Ah	C5h	CAPCOM Register 9 Interrupt Control Register	0000h
CC10IC	b FF8Ch	C6h	CAPCOM Register 10 Interrupt Control Register	0000h
CC11IC	b FF8Eh	C7h	CAPCOM Register 11 Interrupt Control Register	0000h
CC12IC	b FF90h	C8h	CAPCOM Register 12 Interrupt Control Register	0000h
CC13IC	b FF92h	C9h	CAPCOM Register 13 Interrupt Control Register	0000h
CC14IC	b FF94h	CAh	CAPCOM Register 14 Interrupt Control Register	0000h
CC15IC	b FF96h	CBh	CAPCOM Register 15 Interrupt Control Register	0000h
ADCIC	b FF98h	CCh	A/D Converter End of Conversion Interrupt Control Register	0000h
ADEIC	b FF9Ah	CDh	A/D Converter Overrun Error Interrupt Control Reg	0000h
T0IC	b FF9Ch	CEh	CAPCOM Timer 0 Interrupt Control Register	0000h
T1IC	b FF9Eh	CFh	CAPCOM Timer 1 Interrupt Control Register	0000h
ADCON	b FFA0h	D0h	A/D Converter Control Register	0000h
P5	b FFA2h	D1h	Port5 Register (read only)	XXXXh
TFR	b FFACh	D6h	Trap Flag Register	0000h
WDTCN	FFAEh	D7h	Watchdog Timer Control Register	000xh
S0CON	b FFB0h	D8h	Serial Channel 0 Control Register	0000h
SSCCN	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 FFCCCh	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

## 20.5 - Special Notes

### PEC Pointer Registers

The source and destination pointers for the peripheral event controller are mapped to a special area within the internal RAM. Pointers that are not occupied by the PEC may therefore be used like normal RAM. During Power Down mode or any 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.

## 20.6 - Identification Registers

The ST10F167 does not have identification registers. The ST10C167 and ST10R167 have four Identification registers, mapped in ESFR space. These registers contain:

- A manufacturer identifier.
- A chip identifier with its revision.
- A internal ROM / Flash and size identifier.
- Programming voltage description.

### IDMANUF (F07Eh / 3Fh) ESFR Reset Value: 0400h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
<b>MANUF</b>												0	0	0	0	0
R																

Bit	Function
MANUF	<b>Manufacturer Identifier</b> 0400h: STMicroelectronics manufacturer (JTAG worldwide normalization).

### IDCHIP (F07Ch / 3Eh) ESFR Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>IDCHIP</b>															
R															

Bit	Function
IDCHIP	<b>Device Identifier</b> Refer to datasheet for values.

## ST10X167

**IDMEM (F07Ah / 3Dh)**

ESFR

Reset Value: **UUUUh**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>MEMTYP</b>								<b>MEMSIZE</b>							
R								R							

Bit	Function
MEMSIZE	<b>Internal Memory Size</b> Refer to datasheet for values. Internal Memory size is 4 x (MEMSIZE) (in K Byte).
MEMTYP	<b>Internal Memory Type</b> Refer to datasheet for values.

**IDPROG (F078h / 3Ch)**

ESFR

Reset Value: **UUUUh**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>PROGVPP</b>								<b>PROGVDD</b>							
R								R							

Bit	Function
PROGVDD	Programming Vdd Voltage. 00h for ST10C167 and ST10R167 <sup>1</sup>
PROGVPP	Programming Vpp Voltage. 00h for ST10C167 and ST10R167 <sup>1</sup>

Note 1. Not implemented for ST10F167.



## 21 - SYSTEM PROGRAMMING

Constructs for modularity, loops, and context switching have been built into the ST10X167 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 ST10X167.

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 ST10X167:

Substituted Instruction	ST10X167 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 ST10X167 provides a Von-Neumann memory architecture and its on-chip hardware

automatically detects accesses to internal RAM, GPRs, and SFRs.

### Multiplication and Division

Multiplication and division of Words and double Words is provided through multiple cycle instructions implementing a Booth algorithm. Each instruction implicitly uses the 32 Bit register MD (MDL = lower 16 Bit, MDH = upper 16 Bit).

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 16/16Bit 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 16Bit 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 ST10X167. 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.

#### 21.1 - Stack Operations

The ST10X167 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 4K Byte address range.

The size of the physical stack area within the internal RAM that effectively is used for standard stack operations is defined via Bitfield STKSZ in register SYSCON (see below).

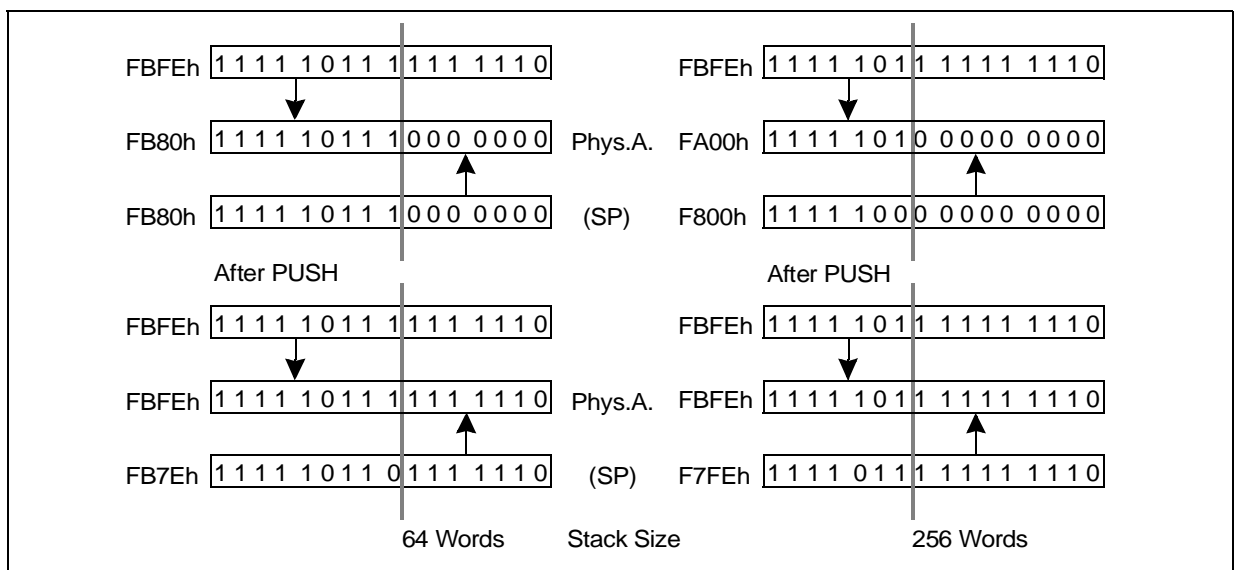
**Table 45 :** Stack Size Selection

(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 1K Byte 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’DFEh...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 151).

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 151 :** 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 = FBFEEh
```

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 (Bitfield STKSZ in register SYSCON).
- Define two pointers, which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.

- Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six Words (for the reserved space to store two interrupt entries).

The internal stack will now fill until the overflow pointer is reached. After entry into the overflow trap procedure, the top of the stack will be copied to the external memory. The internal pointers will then be modified to reflect the newly allocated space. After exiting from the trap procedure, the internal stack will wrap around to the top of the internal stack, and continue to grow until the new value of the stack overflow pointer is reached.

When the underflow pointer is reached while the stack is emptied the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.

### Linear Stack

The ST10X167 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'FD FEh. 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.

### 21.2 - 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.

### 21.3 - Procedure Call Entry and Exit

To support modular programming a procedure mechanism is provided to allow coding of frequently used portions of code into subroutines. The CALL and RET instructions store and restore the value of the instruction pointer (IP) on the system stack before and after a subroutine is executed.

Procedures may be called conditionally with instructions CALLA or CALLI, or be called unconditionally using instructions CALLR or CALLS.

**Note** Any data pushed onto the system stack during execution of the subroutine must be popped before the RET instruction is executed.

### Passing Parameters on the System Stack

Parameters may be passed via the system stack through PUSH instructions before the subroutine is called, and POP instructions during execution of the subroutine. Base plus offset indirect addressing also permits access to parameters without popping these parameters from the stack during execution of the subroutine. Indirect addressing provides a mechanism of accessing data referenced by data pointers, which are passed to the subroutine. In addition, two instructions have been implemented to allow one parameter to be passed on the system stack without additional software overhead.

The PCALL (push and call) instruction first pushes the 'reg' operand and the IP contents onto the system stack and then passes control to the subroutine specified by the 'caddr' operand.

When exiting from the subroutine, the RETP (return and pop) instruction first pops the IP and then the 'reg' operand from the system stack and returns to the calling program.

### Cross Segment Subroutine Calls

Calls to subroutines in different segments require the use of the CALLS (call inter-segment subroutine) instruction. This instruction preserves both the CSP (code segment pointer) and IP on the system stack.

Upon return from the subroutine, a RETS (return from inter-segment subroutine) instruction must be used to restore both the CSP and IP. This ensures that the next instruction after the CALLS instruction is fetched from the correct segment.

**Note** It is possible to use CALLS within the same segment, but still two Words of the stack are used to store both the IP and CSP.

### Providing Local Registers for Subroutines

For subroutines which require local storage, the following methods are provided:

**Alternate bank of registers:** Upon entry into a subroutine, it is possible to specify a new set of local registers by executing the SCXT (switch context) instruction. This mechanism does not provide a method to recursively call a subroutine.

**Saving and restoring of registers:** To provide local registers, the contents of the registers which are required for use by the subroutine can be pushed onto the stack and the previous values be popped before returning to the calling routine. This is the most common technique used today and it does provide a mechanism to support recursive procedures. This method, however, requires two instruction cycles per register stored on the system stack (one cycle to PUSH the register, and one to POP the register).

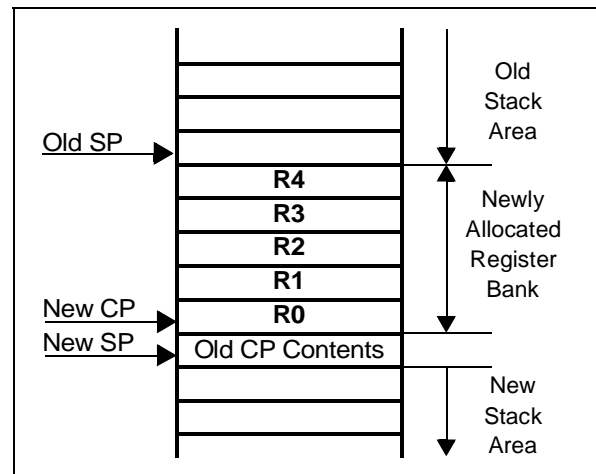
**Use of the system stack for local registers:** It is possible to use the SP and CP to set up local subroutine register frames. This enables subroutines to dynamically allocate local variables as needed within two instruction cycles.

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 register is then accessed as if it

was a normal register. Upon exit from the subroutine, first the old CP must be restored by popping it from the stack and then the number of used local registers must be added to the SP to restore the allocated local space back to the system stack. The system stack is growing downwards, while the register bank is growing upwards.

**Figure 152 :** Local registers



The software to provide the local register bank for the example above Figure 152 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
```

### 21.4 - Table Searching

A number of features have been included to decrease the execution time required to search tables. First, branch delays are eliminated by the branch target cache after the first iteration of the loop. Second, in non-sequentially searched tables, the enhanced performance of the ALU allows more complicated hash algorithms to be processed to obtain better table distribution. For sequentially searched tables, the auto-increment indirect addressing mode and the E (end of table) flag stored in the PSW decrease the number of overhead instructions executed in the loop.

The two examples below illustrate searching ordered tables and non-ordered tables, respectively:

```
      MOV      R0, #BASE      ;Move table base into R0
LOOP:  CMP      R1, [R0+]     ;Compare target to table entry
      JMPR     cc_SGT, LOOP   ;Test whether target has not been found
```

The last entry in the table must be greater than the largest possible target.

```
      MOV      R0, #BASE      ;Move table base into R0
LOOP:  CMP      R1, [R0+]     ;Compare target to table entry
      JMPR     cc_NET, LOOP   ;Test whether target is not found AND the
                               end of table...
                               ;...has not been reached.
```

Note The last entry in the table must be equal to the lowest signed integer (8000h).

### 21.5 - Peripheral Control and Interface

All communication between peripherals and the CPU is performed either by PEC transfers to and from internal memory, or by explicitly addressing the SFRs associated with the specific peripherals. After resetting the ST10X167 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.

### 21.6 - Floating Point Support

All floating point operations are performed using software. Standard multiple precision instructions are used to perform calculations on data types that exceed the size of the ALU. Multiple Bit rotate and logic instructions allow easy masking and extracting of portions of floating point numbers.

To decrease the time required to perform floating point operations, two hardware features have been implemented in the CPU core. First, the PRIOR instruction aids in normalizing floating point numbers by indicating the position of the first set Bit in a GPR. This result can be used to rotate the floating point result accordingly.

The second feature aids in properly rounding the result of normalized floating point numbers through the overflow (V) flag in the PSW. This flag is set when a one is shifted out of the carry Bit during shift right operations. The overflow flag and the carry flag are then used to round the floating point result based on the desired rounding algorithm.



### 21.7 - Trap / Interrupt Entry and Exit

Interrupt routines are entered when a requesting interrupt has a priority higher than the current CPU priority level. Traps are entered regardless of the current CPU priority. When either a trap or interrupt routine is entered, the state of the machine is preserved on the system stack and a branch to the appropriate trap/interrupt vector is made.

All trap and interrupt routines require the use of the RETI (return from interrupt) instruction to exit from the called routine.

This instruction restores the system state from the system stack and then branches back to the location where the trap or interrupt occurred.

### 21.8 - Inseparable Instruction Sequences

The instructions of the ST10X167 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
```

### 21.9 - Overriding the DPP Addressing Mechanism

The standard mechanism to access data locations uses one of the four data page pointers (DPPx), which selects a 16K Byte data page, and a 14 Bit offset within this data page. The four DPPs allow immediate access to up to 64K Byte 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 64K Byte segment oriented data access scheme for 1...4 instructions without having to change the current DPPs. In this case all 16 Bit 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 ST10X167 (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 EXTSTR instructions combine the DPP override mechanism with the redirection to the ESFR space using a single instruction.

Note Instructions EXTR, EXTPR and EXTSTR 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.

### 21.10 - Handling the Internal ROM

The mask ROM or Flash Memory versions of the ST10X167 may provide and control a 32K Byte internal ROM area that may store code as well as data. Access to this internal ROM area is controlled during the reset configuration and via software. The ROM area may be mapped to segment 0, to segment 1 or may be disabled at all.

Note The internal ROM area always occupies an address area of 32KByte, even if the implemented mask ROM or Flash memory is smaller than that (e.g. 8KByte).

Of course the total implemented memory may exceed 32KBytes.

### ROM Configuration During Reset

The control input pin  $\overline{EA}$  (External Access) enables the user to define the address area from which the first instructions after reset are fetched. When  $\overline{EA}$  is low ('0') during reset, the internal ROM area is disabled and the first instructions are fetched from external memory.

When  $\overline{EA}$  is high ('1') during reset, the internal ROM area is globally enabled and the first instructions are fetched from the internal ROM.

Note DO NOT select internal ROM access after reset on ROMless devices.

### Mapping the Internal ROM Area

After reset the internal ROM area is mapped into segment 0, the “system segment” (00’0000h...00’7FFFh) as a default. This is necessary to allow the first instructions to be fetched from locations 00’0000h. The ROM area may be mapped to segment 1 (01’0000h...01’7FFFh) by setting Bit ROMS1 in register SYSCON.

The internal ROM may now be accessed through the lower half of segment 1, while accesses to segment 0 will now be made to external memory. This adds flexibility to the system software.

The interrupt/trap vector table, which uses locations 00’0000h through 00’01FFh, is now part 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 ROM 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 ROM Area After Reset

If the internal ROM does not contain an appropriate start-up code, the system may be booted from external memory, while the internal ROM is enabled afterwards to provide access to library routines, tables, etc.

If the internal ROM only contains the start-up code and/or test software, the system may be booted from internal ROM, 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 ROM area, which is now unnecessary.

#### 21.11 - 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 ST10X167 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 ROM access after reset:** When the first instructions are to be fetched from internal ROM (EA=’1’), the memory must contain a valid reset vector and valid code at its destination.

**Mapping the internal ROM to segment 1:** Due to instruction pipelining, any new ROM mapping will at the earliest become valid for the second instruction after the instruction which has changed the ROM mapping. To enable accesses to the ROM after mapping a branch to the newly selected ROM area (JMPS) and reloading of all data page pointers is required.

This also applies to re-mapping the internal ROM to segment 0.

**Enabling the internal ROM after reset:** When enabling the internal ROM after having booted the system from external memory, note that the ST10X167 will then access the internal ROM using the current segment offset, rather than accessing external memory.

**Disabling the internal ROM after reset:** When disabling the internal ROM after having booted the system from there, note that the ST10X167 will not access external memory before a jump to segment 0 (in this case) is executed.

### General Rules

When mapping the ROM no instruction or data accesses should be made to the internal ROM, otherwise unpredictable results may occur.

To avoid these problems, the instructions that configure the internal ROM should be executed from external memory or from the internal RAM.

Whenever the internal ROM is disabled, enabled or re-mapped the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal ROM and/or external memory.

## 22 - KEY WORD INDEX

Symbols		C
(	249	
<b>A</b>		
Acronyms	12	
Adapt Mode	250	
ADC	24, 211	
ADCON	212	
Address		
Arbitration	122	
Area Definition	121	
Boundaries	33	
Segment	110, 251	
ADDRSELx	121, 122	
ALE length	112	
ALU	44	
Analog/Digital Converter	24, 211	
Arbitration		
Address	122	
External Bus	124	
ASC0		
Interrupts	163	
Auto Scan conversion	214	
<b>B</b>		
Baudrate		
ASC0	162	
Bootstrap Loader	182	
CAN	226	
SSC	174	
BHE	88, 110	
Bit		
addressable memory	29	
Handling	40	
protected	40	
timing register	227	
Bootstrap Loader	179, 250	
Boundaries	33	
Burst mode (PWM)	205	
Bus		
Arbitration	124	
CAN	22, 220, 240	
Demultiplexed	107	
Idle State	123	
Mode Configuration	106, 250	
Multiplexed	106	
BUSCONx	119	
CAN Interface	22, 220	
CAPCOM	23	
interrupt	198	
timer	186	
unit	183	
Capture mode	191	
Capture Mode (GPT)	141, 151	
Capture/Compare unit	183	
CCM0, CCM1, CCM2, CCM3	190	
CCM4, CCM5, CCM6, CCM7	190	
Center aligned PWM	204	
Chip Select	110, 251	
Clock Generator	251	
Compare modes	192	
Concatenation of Timers	139, 151	
Configuration		
Address	110, 251	
Bus Mode	106, 250	
Chip Select	110, 251	
PLL	251	
Reset	246	
Write Control	250	
Context Switching	66	
Control / Status Register	223	
Conversion		
analog/digital	211	
Auto Scan	214	
timing control	218	
Count direction	131, 146	
Counter	132, 138, 148, 150, 207	
CP	49	
CPU	13	
CRIC	154	
CSP	46	
<b>D</b>		
Data Page	48, 281	
boundaries	33	
Delay		
Read/Write	114	
Demultiplexed Bus	107	
Direction		
count	131, 146	
Disable		
Interrupt	64	



Message Control Register .....229

Multiplexed Bus .....106

Multiplication ..... 52, 273

**N**

NMI ..... 55, 73

**O**

ODP2 .....82

ODP3 .....85

ODP6 .....93

ODP7 .....96

ODP8 .....99

ONES .....54

Open Drain Mode .....74

**P**

P0L,P0H .....77

P1L, P1H .....249

P2 .....82

P3 .....85

P4 .....88

P5 .....90

P6 .....92

P7 .....95

P8 .....99

PEC ..... 16, 17, 31, 62

Response Times .....67

PECCx .....62

Peripheral .....21

PICON .....76

Pins .....102

in Idle and Power Down mode .....256

Pipeline .....35

Effects .....37

PLL .....251

Port .....22

input threshold .....75

Power Down Mode .....253

Protected

Bits .....40

PSW ..... 43, 61

Pulse Width Modulation .....24

PWM .....24

PWM Module .....200

PWMCON0 .....208

PWMCON1 .....209

**R**

RAM

extension .....32

internal .....28

Read/Write Delay .....114

READY .....115

Register .....257, 259, 265

Reset .....242

Configuration .....246

Output .....245

Values .....246

ROM .....282

RP0H .....249

**S**

S0BG .....162

S0CON .....157

S0RBUF ..... 161, 162

S0RIC .....163

S0TBIC, S0EIC .....164

S0TBUF ..... 160, 162

S0TIC .....163

Segment

Address ..... 110, 251

boundaries .....33

Segmentation

Enable/Disable .....43

Serial Interface .....22

Asynchronous .....158

CAN ..... 22, 220

Synchronous ..... 161, 165

SFR .....31, 259, 265

Single Chip Mode .....104

Single shot mode (PWM) .....206

Slave mode .....124

Software

Reset .....242

Traps .....71

Source

Interrupt .....56

SP .....51

SSC .....165

Baudrate generation .....174

Error Detection .....174

Full Duplex .....170

Half Duplex .....172

SSCBR .....174

SSCCON .....	167	T7 .....	186
SSCEIC .....	176	T78CON .....	186
SSCRB, SSCTB .....	172	T8 .....	186
SSCRIC .....	176	Threshold .....	75
SSCTIC .....	176	Timer .....	23, 128, 143
Stack .....	29, 51, 275	Auxiliary Timer .....	137, 149
Startup Configuration .....	246	CAPCOM .....	186
STKOV .....	51	Concatenation .....	139, 151
STKUN .....	52	Core Timer .....	130, 145
Subroutine .....	278	Traps .....	58, 71
Synchronous Serial Interface .....	165	Tri-State Time .....	113
SYSCON .....	42		
System Reset		<b>U</b>	
Startup Configurations .....	247	Unseparable instructions .....	281
<b>T</b>		Upper Arbitration Reg .....	230
T0 .....	186	Upper Global Mask Long .....	227
T01CON .....	186	Upper Mask of Last Message .....	228
T1 .....	186	<b>W</b>	
T2CON .....	137	Waitstate	
T2IC .....	142	Memory Cycle .....	113
T2IC, T3IC, T4IC .....	142	Tri-State .....	113
T3CON .....	130	Watchdog .....	23, 177, 245
T3IC .....	142	WDTCON .....	177
T4CON .....	137	<b>X</b>	
T4IC .....	142	XBUS .....	18, 127
T5CON .....	149	XRAM on-chip .....	32
T5IC .....	154	<b>Z</b>	
T6CON .....	145	ZEROS .....	54
T6IC .....	154		

## 23 - INDEX OF REGISTERS

ADCIC (FF98h / CCh)	SFR	F167 Reset Value: 0XX0h	42
ADCON (FFA0h / D0h)	SFR	Reset Value: --00h	219
ADDAT (FEA0h / 50h)	SFR	Reset Value: 0000h	212
ADDAT2 (F0A0h / 50h)	SFR	Reset Value: 0000h	213
ADDRSEL1 (FE18h / 0Ch)	ESFR	Reset Value: 0000h	213
ADDRSEL2 (FE1Ah / 0Dh)	SFR	Reset Value: 0000h	120
ADDRSEL3 (FE1Ch / 0Eh)	SFR	Reset Value: 0000h	120
ADDRSEL4 (FE1Eh / 0Fh)	SFR	Reset Value: 0000h	121
ADEIC (FF9Ah / CDh)	SFR	Reset Value: 0000h	121
Bit Timing Register (EF04h)	SFR	Reset Value: --00h	219
BUSCON0 (FF0Ch / 86h)	XReg	Reset Value: UUUUh	227
BUSCON1 (FF14h / 8Ah)	SFR	Reset Value: 0XX0h	119
BUSCON2 (FF16h / 8Bh)	SFR	Reset Value: 0000h	119
BUSCON3 (FF18h / 8Ch)	SFR	Reset Value: 0000h	119
BUSCON4 (FF1Ah / 8Dh)	SFR	Reset Value: 0000h	119
CCM0 (FF52h / A9h)	SFR	Reset Value: 0000h	190
CCM1 (FF54h / AAh)	SFR	Reset Value: 0000h	190
CCM2 (FF56h / ABh)	SFR	Reset Value: 0000h	190
CCM3 (FF58h / ACh)	SFR	Reset Value: 0000h	190
CCM4 (FF22h / 91h)	SFR	Reset Value: 0000h	190
CCM5 (FF24h / 92h)	SFR	Reset Value: 0000h	190
CCM6 (FF26h / 93h)	SFR	Reset Value: 0000h	190
CCM7 (FF28h / 94h)	SFR	Reset Value: 0000h	190
CCxIC (see Table 32)	SFR/ESF	Reset Value: --00h	198
Control / Status Register (EF00h)	XReg	Reset Value: XX01h	223
CP (FE10h / 08h)	SFR	Reset Value: FC00h	49
CRIC (FF6Ah / B5h)	SFR	Reset Value: --00h	154
CSP (FE08h / 04h)	SFR	Reset Value: 0000h	46
DP0H (F102h / 81h)	SFR	Reset Value: -- 00h	78
DP0L (F100h / 80h)	ESFR	Reset Value: -- 00h	78
DP1H (F106h / 83h)	ESFR	Reset Value: -- 00h	80
DP1L (F104h / 82h)	ESFR	Reset Value: -- 00h	80
DP2 (FFC2h / E1h)	SFR	Reset Value: 0000h	82
DP3 (FFC6h / E3h)	SFR	Reset Value: 0000h	85
DP4 (FFCAh / E5h)	SFR	Reset Value: -- 00h	89
DP6 (FFCEh / E7h)	SFR	Reset Value: -- 00h	92
DP7 (FFD2h / E9h)	SFR	Reset Value: -- 00h	96
DP8 (FFD6h / EBh)	SFR	Reset Value: -- 00h	99
DPP0 (FE00h / 00h)	SFR	Reset Value: 0000h	48
DPP1 (FE02h / 01h)	SFR	Reset Value: 0001h	48
DPP2 (FE04h / 02h)	SFR	Reset Value: 0002h	48
DPP3 (FE06h / 03h)	SFR	Reset Value: 0003h	48
EXICON (F1C0h / E0h)	ESFR	Reset Value: 0000h	70
EXICON (F1C0h / E0h)	ESFR	Reset Value: 0000h	254
Global Mask Short (EF06h)	XReg	Reset Value: UFUUh	227
IDCHIP (F07Ch / 3Eh)	ESFR	Reset Value: UUUUh	271
IDMANUF (F07Eh / 3Fh)	ESFR	Reset Value: 0400h	271
IDMEM (F07Ah / 3Dh)	ESFR	Reset Value: UUUUh	272



IDPROG (F078h / 3Ch)	ESFR	Reset Value: UUUUh	272
Interrupt Register (EF02h)	XReg	Reset Value: --XXh	225
IP (---- / --)	---	Reset Value: 0000h	46
Lower Arbitration Reg (EFn4h)	XReg	Reset Value: UUUUh	230
Lower Global Mask Long (EF0Ah)	XReg	Reset Value: UUUUh	228
Lower Mask of Last Message (EF0Eh)	XReg	Reset Value: UUUUh	228
MDC (FF0Eh / 87h)	SFR	Reset Value: 0000h	53
MDH (FE0Ch / 06h)	SFR	Reset Value: 0000h	52
MDL (FE0Eh / 07h)	SFR	Reset Value: 0000h	53
Message Configuration Register (EFn6h)	XReg	Reset Value: --UUh	231
Message Control Register (EFn0h)	XReg	Reset Value: UUUUh	229
ODP2 (F1C2h / E1h)	ESFR	Reset Value: 0000h	82
ODP3 (F1C6h / E3h)	ESFR	Reset Value: 0000h	85
ODP6 (F1CEh / E7h)	ESFR	Reset Value: - - 00h	93
ODP7 (F1D2h / E9h)	ESFR	Reset Value: - - 00h	96
ODP8 (F1D6h / EBh)	ESFR	Reset Value: - - 00h	99
ONES (FF1Eh / 8Fh)	SFR	Reset Value: FFFFh	54
P0H (FF02h / 81h)	SFR	Reset Value: - - 00h	77
P0L (FF00h / 80h)	SFR	Reset Value: - - 00h	77
P1H (FF06h / 83h)	SFR	Reset Value: - - 00h	80
P1L (FF04h / 82h)	SFR	Reset Value: - - 00h	80
P2 (FFC0h / E0h)	SFR	Reset Value: 0000h	82
P3 (FFC4h / E2h)	SFR	Reset Value: 0000h	85
P4 (FFC8h / E4h)	SFR	Reset Value: - - 00h	88
P5 (FFA2h / D1h)	SFR	Reset Value: XXXXh	90
P6 (FFCCh / E6h)	SFR	Reset Value: - - 00h	92
P7 (FFD0h / E8h)	SFR	Reset Value: - - 00h	95
P8 (FFD4h / EAh)	SFR	Reset Value: - - 00h	99
PECCx (FECyh / 6zh, see Table 8)	SFR	Reset Value: 0000h	62
PICON (F1C4h / E2h)	ESFR	Reset Value: - - 00h	76
PSW (FF10h / 88h)	SFR	Reset Value: 0000h	61
PSW (FF10h / 88h)	SFR	Reset Value: 0000h	44
PWMCON0 (FF30h / 98h)	SFR	Reset Value: 0000h	208
PWMCON1 (FF32h / 99h)	SFR	Reset Value: 0000h	209
PWMIC (F17Eh / BFh)	ESFR	Reset Value: --00h	209
REG_NAME (A16h / A8h)	SFR/ESFR/XReg	Reset Value: ****h	257
REG_NAME (A16h / A8h)	SFR/ESFR/XReg	Reset Value: --**h	257
RP0H (F108h / 84h)	SFR	Reset Value: - - XXh	122
RP0H (F108h / 84h)	SFR	Reset Value: --XXh	249
S0CON (FFB0h / D8h)	SFR	Reset Value: 0000h	157
S0EIC (FF70h / B8)	SFR	Reset Value: - - 00h	164
S0RIC (FF6Eh / B7h)	SFR	Reset Value: - - 00h	163
S0TBIC (F19Ch / CEh)	ESFR	Reset Value: - - 00h	164
S0TIC (FF6Ch / B6h)	SFR	Reset Value: - - 00h	163
SP (FE12h / 09h)	SFR	Reset Value: FC00h	51
SSCCON (FFB2h / D9h)	SFR	Reset Value: 0000h	168
SSCCON (FFB2h / D9h)	SFR	Reset Value: 0000h	169
SSCEIC (FF76h / BBh)	SFR	Reset Value: --00h	176
SSCRIC (FF74h / BAh)	SFR	Reset Value: --00h	176
SSCTIC (FF72h / B9h)	SFR	Reset Value: --00h	176

## ST10X167

---

STKOV (FE14h / 0Ah)	SFR	Reset Value: FA00h	51
STKUN (FE16h / 0Bh)	SFR	Reset Value: FC00h	52
SYSCON (FF12h / 89h)	SFR	C/R167 Reset Value: 0X00h	42
SYSCON (FF12h / 89h)	SFR	Reset Value: 0X00h 1	118
SYSCON (FF12h / 89h)	SFR	Reset Value: 0X00h 1	253
T01CON (FF50h / A8h)	SFR	Reset Value: 0000h	187
T0IC (FF9Ch / CEh)	SFR	Reset Value: --00h	189
T1IC (FF9Eh / CFh)	SFR	Reset Value: --00h	189
T2CON (FF40h / A0h)	SFR	Reset Value: 0000h	137
T2IC (FF60h / B0h)	SFR	Reset Value: - - 00h	142
T3CON (FF42h / A1h)	SFR	Reset Value: 0000h	130
T3IC (FF62h / B1h)	SFR	Reset Value: - - 00h	142
T4CON (FF44h / A2h)	SFR	Reset Value: 0000h	137
T4IC (FF64h / B2h)	SFR	Reset Value: - - 00h	142
T5CON (FF46h / A3h)	SFR	Reset Value: 0000h	149
T5IC (FF66h / B3h)	SFR	Reset Value: --00h	154
T6CON (FF48h / A4h)	SFR	Reset Value: 0000h	145
T6IC (FF68h / B4h)	SFR	Reset Value: --00h	154
T78CON (FF20h / 90h)	SFR	Reset Value: 0000h	187
T7IC (F17Ah / BEh)	ESFR	Reset Value: --00h	189
T8IC (F17Ch / BFh)	ESFR	Reset Value: --00h	189
TFR (FFACh / D6h)	SFR	Reset Value: 0000h	72
Upper Arbitration Reg (EFn2h)	XReg	Reset Value: UUUUh	230
Upper Global Mask Long (EF08h)	XReg	Reset Value: UUUUh	227
Upper Mask of Last Message (EF0Ch)	XReg	Reset Value: UUUUh	228
WDTCN (FFAEh / D7h)	SFR	Reset Value: 000Xh	178
xxIC (yyyyh / zzh)	SFR Area	Reset Value: 00h	59
ZEROS (FF1Ch / 8Eh)	SFR	Reset Value: 0000h	54

## 24 - REVISION HISTORY

### 24.1 - Revision of the 28th of August 2000

This is revision 1.1 of this document, released on 28th of August 2000. The differences between previous revisions and revision 1.1 are:

- page 6, Table of content, chap18 (Previous revision 18.1 to 18.5 / New revision 18.1 to 18.6) reflects the modifications of the text
- page 10, Abbreviations IRAM on-chip Internal RAM added
- page 11, figure 1, XTAL1-XTAL2, RxD-TxD added
- page 12, figure 2 up date block diagram with XRAM
- page 17, figure 3 updated with POH.7, POH.6, POH.5 pins, table 1, addition of footnote
- page 18, chapter 2.4.1: Vdd ( all Vcc changed to Vdd)
- page 24, chapter 3, figure 4 updated
- page 26, figure 6 updated
- page 30, figure 8 RAM/SFR area changed from 00'F000h to 00'F600h
- page 40, chapter 4.4.1 Syscon bit OWDDIS PLL base frequency: 2 to 10 MHz, WRCFG bit text updated
- page 47, figure 15 updated, CP register text "Do not set CP below IRAM start address 00'F600h (2K Byte)
- page 50, chapter 4.4.12, MDC register, bit MS added
- page 55, chapter 5.1., order of comments and note of table6 are inverted
- page 82, figure 31, updated
- page 83, chapter 6.5 text ODP2 changed to ODP3
- page 84, figure 32 pin WRH changed to WRH,
- page 91, chapter 6.8.1, figure 39 updated
- page 96, figure 44, read buffer, Read P2.y changed to Read P7.y
- page 120, figure 59 updated
- page 121, RPOH, Bit WRCFG changed to WRC and "0" is normal state is replaced to "1" and vice versa
- page 134, correction of figure 71
- page 148, correction of figure 82
- page 150, figure 84 addition of T3IN and T3EUD input pins
- page 154, figure 87 bit 11 of S0CON register removed
- page 164, figure 93 bit 7 and 13 of SSCON register removed
- page 175, figure 99 bits 2 to 7 of WDTCON removed
- page 178, figure 102 R<sub>P0L.4</sub> changed to 8 K $\Omega$  maximum
- page 183, figure 106 x = 0, 7 and y = 1, 8 added
- page 188, CC16..CC32 changed to CC16..CC31
- page 209, figure 123 data bit 10 and 11 removed from ADDAT and ADDAT2 and bit 6 removed from ADCON
- page 222, CAN control status register name of bit test changed to TST
- page 224, table 36, update of notes
- page 227, message control register bit (MSGLST) changed to (MSGLST CPUUPD)
- page 234, figure 136, INTPNDd changed to INTPND
- page 240, chapter 18.1.1, 18.1.2 and 18.1.3 updated
- page 246, table 39 redrawn, table 40 added
- page 248, bootstrap loader acknowledge Byte C5h added
- page 251, SSPEN (bit 2 of SYSCON) changed to XPEN
- page 253, figure 148 update of pull-down current (200 $\mu$ A)
- page 262, table 44 notes removed
- page 269, correction of identification registers
- page 274, table 45 stack size updated for STKSZ = 111b
- page 276, user stack Rb, [Rw+] or Rw, [Rw+] text changed
- pages 282 to 285, update of KEY WORD INDEX

– page 289, update revision history

### **24.2 - Revision of the 7th of August 2002**

– pages 1, 2 and 294 (cover and last page) have been inserted in the document numbering leading to a total of 294 pages instead of 290.

– page 130, bits T3M of T3CON register, combination 1XX (Reserved. Do not use this combination) has been changed to 111 .

– page 149, bit CT3 of T5CON register is bit 10 instead of bit 9 and its functionality when 0 is "Capture triggered from CAPIN pin".

End of file - 7th of August 2002



Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is a registered trademark of STMicroelectronics

© 2002 STMicroelectronics All Rights Reserved

STMicroelectronics GROUP OF COMPANIES

Australia - Brazil - Canada - China - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco  
Singapore - Spain - Sweden - Switzerland - United Kingdom - United States

<http://www.st.com>