UM0404
User manual

High performance ST10F276Z5 and ST10F276E 16-bit microcontrollers

Introduction

This manual describes the functionality of the ST10F276Z5 and ST10F276E devices.

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 sections.

The explanation of memory configuration has been restricted to that of the internal addressable memory space. The ST10F276Z5 and ST10F276E Flash configurations are not discussed in this manual. Refer to the ST10F276Z5 and ST10F276E datasheets for detailed information (see *Appendix B: Document references*).

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 *Section 27: System programming on page 525*.

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 datasheets.

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

The ST10F276Z5 and ST10F276E software and hardware development tools include:

- Compilers (C, C++), macro-assemblers, linkers, locators, library managers, format-converters from Tasking & Keil
- HLL debuggers
- Real-time operating systems
- In-circuit emulators (based on Bond-out ST chips) from Hitex, Lauterbach, Nohau
- Logic analyzer disassemblers
- Evaluation boards with monitor programs from FORTH
- Industrial embedded Flash programming software from PLS
- Network driver software (CAN)

Contents

1 Architectural overview 22

1.1 Basic CPU concepts and optimization 22

1.1.1 High instruction bandwidth / fast execution 24

1.1.2 High function 8-bit and 16-bit ALU 25

1.1.3 Extended bit processing and peripheral control 25

1.1.4 High performance branch, call and loop processing 25

1.1.5 Consistent and optimized instruction formats 26

1.1.6 Programmable multiple priority interrupt system 27

1.2 On-chip system resources 27

1.2.1 Peripheral event control and interrupt control 28

1.2.2 Memory areas 28

1.2.3 External bus interface 29

1.3 Clock generator 30

1.3.1 PLL operation 31

1.3.2 Prescaler operation 31

1.3.3 Direct drive 31

1.3.4 Oscillator watchdog (OWD) 32

1.4 On-chip peripheral blocks 32

1.4.1 Peripheral interfaces 33

1.4.2 Peripheral timing 33

1.4.3 Programming hints 33

1.4.4 Parallel ports 34

1.4.5 Serial channels 34

1.4.6 General purpose timer (GPT) unit 35

1.4.7 Watchdog timer 36

1.4.8 Capture / compare (CAPCOM) units 36

1.4.9 Pulse width modulation unit 37

1.4.10 A/D converter 37

1.4.11 CAN module 38

1.4.12 I2C serial interface 38

1.5 Real time clock 38

1.6 Protected bits 39

2 Memory organization 40

2.1 Word, byte and bit storage 41
2.2 On-chip Flash 42
2.3 IRAM and SFR area 44
2.3.1 System stack 46
2.3.2 General purpose registers 46
2.3.3 PEC source and destination pointers 47
2.3.4 Special function registers 48
2.4 The on-chip XRAM 49
2.4.1 XRAM access via external masters 50
2.5 External memory space 50
2.6 Crossing memory boundaries 51

3 The central processing unit (CPU) 52
3.1 Instruction pipelines 54
3.1.1 Sequential instruction processing 55
3.1.2 Standard branch instruction processing 55
3.1.3 Cache jump instruction processing 56
3.1.4 Particular pipeline effects 56
3.2 Bit-handling and bit-protection 59
3.3 Instruction execution times 60
3.4 CPU special function registers 61
3.4.1 The system configuration register SYSCON 62
3.4.2 X-Peripherals control register (XPERCON) 65
3.4.3 XPERCON and XPEREMU registers 66
3.4.4 Emulation dedicated registers 67
3.4.5 The processor status word PSW 67
3.4.6 The instruction pointer IP 70
3.4.7 The code segment pointer CSP 71
3.4.8 The data page pointers DPP0, DPP1, DPP2, DPP3 72
3.4.9 The context pointer CP 73
3.4.10 The stack pointer SP 76
3.4.11 The stack overflow pointer STKOV 76
3.4.12 The stack underflow pointer STKUN 77
3.4.13 The multiply / divide high register MDH 78
3.4.14 The multiply / divide low register MDL 78
3.4.15 The multiply / divide control register MDC 79

3.4.16 The constant zeros register ZEROS 79
3.4.17 The constant ones register ONES 81

4 Multiply-accumulate unit (MAC) 82
4.1 MAC features 82
4.2 MAC operation 83
4.2.1 Instruction pipelining 84
4.2.2 Particular pipeline effects with the MAC unit 84
4.2.3 Address generation 85
4.2.4 16 x 16 signed/unsigned parallel multiplier 86
4.2.5 40-bit signed arithmetic unit 86
4.2.6 The 40-bit signed accumulator register 87
4.2.7 The 40-bit adder / subtracter 88
4.2.8 Data limiter 88
4.2.9 The accumulator shifter 89
4.2.10 Repeat unit 89
4.2.11 MAC interrupt 90
4.2.12 Number representation & rounding 90
4.3 MAC register set 91
4.3.1 Address registers 91
4.3.2 Accumulator & control registers 91
4.4 MAC instruction set summary 94

5 Interrupt and trap functions 96
5.1 Interrupt system structure 96
5.1.1 Normal interrupt processing and PEC service 100
5.1.2 Interrupt system register description 100
5.1.3 Interrupt control registers 100
5.1.4 Interrupt priority level and group level 101
5.1.5 Interrupt control functions in the PSW 103
5.2 Operation of the PEC channels 104
5.3 Prioritizing interrupt & PEC service requests 106
5.3.1 Enabling and disabling interrupt requests 106
5.3.2 Interrupt class management 107
5.4 Saving the status during interrupt service 108
5.4.1 Context switching 109

5.5 Interrupt response times 110
5.5.1 PEC response times 111
5.6 External interrupts 113
5.6.1 Fast external interrupts 115
5.7 X-peripheral interrupt 117
5.8 Trap functions 128
5.8.1 Software traps 129
5.8.2 Hardware traps 129
5.8.3 External NMI trap 131
5.8.4 Stack overflow trap 131
5.8.5 Stack underflow trap 132
5.8.6 Undefined opcode trap 132
5.8.7 MAC interrupt 132
5.8.8 Protection fault trap 132
5.8.9 Illegal word operand access trap 133
5.8.10 Illegal instruction access trap 133
5.8.11 Illegal external bus access trap 133

6 Parallel ports 134

6.1 Introduction 134
6.1.1 Open drain mode 134
6.1.2 Input threshold control 137
6.1.3 Alternate port functions 138
6.2 PORT0 139
6.2.1 Alternate functions of PORT0 140
6.3 PORT1 142
6.3.1 Alternate functions of PORT1 143
6.3.2 PORT1 analog inputs disturb protection 144
6.4 Port2 146
6.4.1 Alternate functions of Port2 147
6.4.2 External interrupts 148
6.5 Port3 150
6.5.1 Alternate functions of Port3 151
6.6 Port4 154
6.6.1 Alternate functions of Port4 155
6.7 Port5 161

6.7.1 Alternate functions of Port5 . . . 162
6.7.2 Port5 analog inputs disturb protection . . . 163
6.8 Port6 . . . 164
6.8.1 Alternate functions of Port6 . . . 165
6.9 Port7 . . . 169
6.9.1 Alternate functions of Port7 . . . 169
6.10 Port8 . . . 172
6.10.1 Alternate functions of Port8 . . . 174

7 Dedicated pins . . . 179

8 The external bus interface . . . 181
8.1 Single chip mode . . . 181
8.2 External bus modes . . . 183
8.2.1 Multiplexed bus modes . . . 183
8.2.2 De-multiplexed bus modes . . . 184
8.2.3 Switching between the bus modes . . . 185
8.2.4 External data bus width . . . 186
8.2.5 Disable / enable control for pin BHE (BYTDIS) . . . 187
8.2.6 Segment address generation . . . 187
8.2.7 CS signal generation . . . 188
8.2.8 Segment address versus chip select . . . 189
8.3 Programmable bus characteristics . . . 190
8.3.1 ALE length control . . . 190
8.3.2 Programmable memory cycle time . . . 191
8.3.3 Programmable memory tri-state time . . . 192
8.3.4 Read / write signal delay . . . 193
8.3.5 READY polarity . . . 194
8.3.6 READY / READY controlled bus cycles . . . 194
8.3.7 Programmable chip select timing control . . . 196
8.4 Controlling the external bus controller . . . 196
8.4.1 Definition of address areas . . . 201
8.4.2 Address window arbitration . . . 201
8.4.3 Precautions and hints . . . 203
8.5 EBC idle state . . . 203
8.6 External bus arbitration . . . 204

8.6.1 Connecting bus masters 204
8.6.2 Entering the hold state 205
8.6.3 Exiting the hold state 206
8.7 The XBUS interface 207
8.8 EA functionality 213

9 The general purpose timer units 215
9.1 Timer block GPT1 215
9.1.1 GPT1 core timer T3 217
9.1.2 GPT1 auxiliary timers T2 and T4 224
9.1.3 Interrupt control for GPT1 timers 231
9.2 Timer block GPT2 232
9.2.1 GPT2 core timer T6 234
9.2.2 Interrupt control for GPT2 timers and CAPREL 244

10 Asynchronous / synchronous serial interface 246
10.1 Asynchronous operation 249
10.2 Synchronous operation 251
10.3 Hardware error detection 253
10.4 ASC0 baud rate generation 253
10.5 ASC0 interrupt control 254

11 XBUS asynchronous / synchronous serial interface 257
11.1 Asynchronous operation 261
11.2 Synchronous operation 264
11.3 Hardware error detection 266
11.4 XASC baud rate generation 266
11.5 XASC interrupt control 267

12 High-speed synchronous serial interface 270
12.1 Full-duplex operation 275
12.2 Half duplex operation 278
12.2.1 Port control 279
12.3 Baud rate generation 280
12.4 Error detection mechanisms 281

12.5 SSC interrupt control 282

13 XBUS high-speed synchronous serial interface 284

13.1 Full-duplex operation 290
13.2 Half duplex operation 293
13.2.1 Port control 294
13.3 Baud rate generation 294
13.4 Error detection mechanisms 295
13.5 XSSC interrupt control 296

14 Watchdog timer 297

14.1 Operation of the watchdog timer 298

15 The bootstrap loader 302

15.1 Selection among user-code, standard or alternate bootstrap 302
15.2 Standard bootstrap loader 303
15.2.1 Entering the standard bootstrap loader 303
15.2.2 ST10 configuration in BSL 305
15.2.3 Booting steps 305
15.2.4 Hardware to activate BSL 306
15.2.5 Memory configuration in bootstrap loader mode 307
15.2.6 Loading the start-up code 308
15.2.7 Exiting bootstrap loader mode 308
15.2.8 Hardware requirements 309
15.3 Standard bootstrap with UART (RS232 or K-Line) 309
15.3.1 Features 309
15.3.2 Entering bootstrap via UART 310
15.3.3 ST10 configuration in UART BSL (RS232 or K-line) 310
15.3.4 Loading the start-up code 311
15.3.5 Choosing the baud rate for the BSL via UART 312
15.4 Standard bootstrap with CAN 313
15.4.1 Features 313
15.4.2 Entering the CAN bootstrap loader 314
15.4.3 ST10 configuration in CAN BSL 315
15.4.4 Loading the start-up code via CAN 315
15.4.5 Choosing the baudrate for the BSL via CAN 316

15.4.6 How to compute the baud rate error . . . 318
15.4.7 Bootstrap via CAN . . . 319
15.5 Comparing the old and the new bootstrap loader . . . 319
15.5.1 Software aspects . . . 319
15.5.2 Hardware aspects . . . 320
15.6 Alternate boot mode (ABM) . . . 320
15.6.1 Activation . . . 320
15.6.2 Memory mapping . . . 320
15.6.3 Interrupts . . . 320
15.6.4 ST10 configuration in alternate boot mode . . . 321
15.6.5 Watchdog . . . 321
15.6.6 Exiting alternate boot mode . . . 321
15.6.7 Alternate boot user software . . . 322
15.6.8 User/alternate mode signature integrity check . . . 322
15.6.9 Alternate boot user software aspects . . . 322
15.6.10 EMUCON register . . . 322
15.6.11 Internal decoding of test modes . . . 323
15.6.12 Example . . . 323
15.7 Selective boot mode . . . 323

16 The capture / compare units . . . 326
16.1 CAPCOM timers . . . 329
16.2 CAPCOM unit timer interrupts . . . 333
16.3 Capture / compare registers . . . 333
16.3.1 Selection of capture modes and compare modes . . . 335
16.4 Capture mode . . . 336
16.5 Compare modes . . . 336
16.5.1 Compare mode 0 . . . 337
16.5.2 Compare mode 1 . . . 338
16.5.3 Compare mode 2 . . . 339
16.5.4 Compare mode 3 . . . 340
16.5.5 Double register compare mode . . . 341
16.6 Capture / compare interrupts . . . 343

17 Pulse width modulation module . . . 345
17.1 Operating modes . . . 347

17.1.1 Mode 0: standard PWM generation (edge aligned PWM) 347
17.1.2 Mode 1: symmetrical PWM generation (center aligned PWM) 348
17.1.3 Burst mode 349
17.1.4 Single shot mode 350
17.2 PWM module registers 351
17.3 Interrupt request generation 354
17.4 PWM output signals 355

18 XBUS pulse width modulation module 357
18.1 Operating modes 359
18.1.1 Mode 0: standard PWM generation (edge aligned PWM) 359
18.1.2 Mode 1: symmetrical PWM generation (center aligned PWM) 360
18.1.3 Burst mode 361
18.1.4 Single shot mode 362
18.2 XPWM module registers 363
18.3 Interrupt request generation 367
18.4 XPWM output signals 368

19 Analog / digital converter 370
19.1 Mode selection and operation 372
19.1.1 Fixed channel conversion modes 375
19.1.2 Auto scan conversion modes 376
19.1.3 Wait for ADDAT read mode 377
19.1.4 Channel injection mode 377
19.1.5 ADC power off (ADOFF) 380
19.2 Conversion timing control 381
19.3 A/D converter interrupt control 382
19.4 Calibration 383
19.5 A/D conversion accuracy 383
19.5.1 Total unadjusted error 384
19.5.2 Analog reference pins 385
19.5.3 Analog input pins 385
19.5.4 Example of external network sizing 390

20 I^2C interface 392
20.1 Main features 392

20.2 General description . . . 393
20.2.1 Mode selection . . . 393
20.2.2 Communication flow . . . 393
20.2.3 SDA/SCL line control . . . 394
20.3 Functional description . . . 395
20.3.1 Slave mode . . . 395
20.3.2 Master mode . . . 397
20.4 Interrupts . . . 400
20.5 Register description . . . 403

21 CAN modules . . . 410
21.1 Memory and pin mapping . . . 410
21.1.1 CAN1 mapping . . . 410
21.1.2 CAN2 mapping . . . 410
21.1.3 Register summary . . . 410
21.2 Interrupt . . . 413
21.3 Configuration support . . . 413
21.3.1 Configuration examples . . . 414
21.4 Clock Prescaling . . . 416
21.5 CAN module: functional overview . . . 417
21.6 Block diagram . . . 418
21.7 Operating modes . . . 418
21.7.1 Software initialization . . . 418
21.7.2 CAN message transfer . . . 419
21.7.3 Disabled automatic re-transmission . . . 419
21.7.4 Test mode . . . 420
21.7.5 Silent mode . . . 420
21.7.6 Loop back mode . . . 420
21.7.7 Loop back combined with silent mode . . . 421
21.7.8 Basic mode . . . 422
21.7.9 Software control of pin CAN_TxD . . . 422
21.8 Programmer's model . . . 422
21.8.1 Hardware reset description . . . 424
21.8.2 CAN protocol related registers . . . 424
21.8.3 Message interface register sets . . . 429
21.8.4 Message handler registers . . . 441

21.9 CAN application 444
21.9.1 Management of message objects 444
21.9.2 Message handler state machine 445
21.9.3 Configuration of a transmit object 448
21.9.4 Updating a transmit object 448
21.9.5 Configuration of a receive object 448
21.9.6 Handling of received messages 449
21.9.7 Configuration of a FIFO buffer 449
21.9.8 Reception of messages with FIFO buffers 450
21.9.9 Handling of interrupts 451
21.9.10 Configuration of the bit timing 452

22 Real time clock 464
22.1 RTC registers 466
22.1.1 RTCCON: RTC control register 466
22.1.2 RTCPH & RTCPL: RTC prescaler registers 467
22.1.3 RTCDH & RTCDL: RTC divider counters 468
22.1.4 RTCH & RTCL: RTC programmable counter registers 469
22.1.5 RTCAH & RTCAL: RTC alarm registers 469
22.2 Programming the RTC 470

23 System reset 472
23.1 Input filter 472
23.2 Asynchronous reset 472
23.3 Synchronous reset (warm reset) 477
23.4 Software reset 483
23.5 Watchdog timer reset 484
23.6 Bidirectional reset 486
23.7 Reset circuitry 490
23.8 Reset application examples 492
23.9 Reset summary 495
23.9.1 System start-up configuration 496

24 Power reduction modes 502
24.1 Idle mode 503
24.2 Power down mode 504

24.2.1 Protected power down mode 506
24.2.2 Interruptible power down mode 506
24.3 Stand-by mode 509
24.3.1 Entering stand-by mode 510
24.3.2 Exiting stand-by mode 511
24.3.3 Real time clock and stand-by mode 511
24.4 Output pin status 511

25 Programmable output clock divider 514

26 Register set 515
26.1 Register description format 515
26.2 General purpose registers (GPRs) 516
26.3 Special function registers ordered by name 517
26.4 Special function registers ordered by address 524
26.5 X-Registers ordered by name 531
26.6 X-Registers ordered by address 536
26.7 Flash registers ordered by name 541
26.8 Flash registers ordered by address 541
26.9 Special notes 542
26.10 Identification registers 543

27 System programming 545
27.1 Stack operations 547
27.2 Register banking 551
27.3 Procedure call entry and exit 551
27.4 Table searching 553
27.5 Peripheral control and interface 554
27.6 Floating point support 554
27.7 Trap / interrupt entry and exit 554
27.8 Inseparable instruction sequences 555
27.9 Overriding the DPP addressing mechanism 555
27.10 Handling the internal Flash 556
27.11 Pits, traps and mines 558

Appendix A Abbreviations 559

Appendix B Document references 560

Revision history 561

List of tables

Table 1. Protected bit . . . 39
Table 2. Segment 8 address range mapping . . . 43
Table 3. 512 Kbyte IFlash memory block organization . . . 43
Table 4. 320 Kbyte XFlash memory block organization . . . 43
Table 5. Stack size . . . 46
Table 6. Mapping of general purpose registers to RAM addresses . . . 47
Table 7. Minimum execution times . . . 61
Table 8. Stack size . . . 64
Table 9. Shift right rounding error evaluation . . . 69
Table 10. Pointer post-modification combinations for IDXi and Rwn . . . 85
Table 11. Parallel data move addressing . . . 86
Table 12. Limiter output using CoSTORE instruction . . . 88
Table 13. MAC register address in CoReg addressing mode . . . 94
Table 14. MAC instruction set summary . . . 94
Table 15. Interrupt and PEC service request sources . . . 97
Table 16. Vector locations and status for hardware traps . . . 99
Table 17. PEC control register addresses . . . 104
Table 18. Example of software controlled interrupt classes . . . 107
Table 19. Pins to be used as external interrupt inputs . . . 114
Table 20. X-Interrupt detailed mapping . . . 118
Table 21. Trap priorities . . . 128
Table 22. Port2 alternate functions . . . 149
Table 23. Port3 alternative functions . . . 152
Table 24. Port4 alternate functions . . . 156
Table 25. Port5 alternate functions . . . 162
Table 26. Port6 alternate functions . . . 166
Table 27. Port7 alternate functions . . . 170
Table 28. Port8 alternate functions . . . 175
Table 29. Summary of dedicated pins . . . 179
Table 30. Definition of address areas . . . 201
Table 31. Status of the external bus interface during EBC idle state . . . 203
Table 32. Definition of XBUS address areas . . . 208
Table 33. GPT1 core timer T3 count direction control . . . 218
Table 34. GPT1 timer resolutions . . . 220
Table 35. GPT1 core timer T3 (counter mode) input edge selection . . . 221
Table 36. GPT1 core timer T3 (incremental interface mode) input edge selection . . . 222
Table 37. GPT1 auxiliary timer (counter mode) input edge selection . . . 226
Table 38. GPT2 core timer T6 count direction control . . . 235
Table 39. GPT2 timer resolution . . . 237
Table 40. GPT2 core timer T6 (counter mode) input edge selection . . . 238
Table 41. GPT2 auxiliary timer (counter mode) input edge selection . . . 240
Table 42. WDTCON bits value on different resets . . . 299
Table 43. WDTREL reload value . . . 300
Table 44. Reset events summary . . . 300
Table 45. ST10F276x boot mode selection . . . 303
Table 46. Ranges of timer contents versus BRP value . . . 317
Table 47. Software topics summary . . . 319
Table 48. Hardware topics summary . . . 320

Table 49. Summary of compare modes . . . 337
Table 50. Register pairs for double-register compare mode. . . . 341
Table 51. CAPCOM unit interrupt control register addresses. . . . 344
Table 52. PWM frequencies . . . 352
Table 53. PWM module channel specific register addresses . . . 353
Table 54. XPWM frequencies . . . 364
Table 55. XPWM module channel specific register addresses. . . . 365
Table 56. ADC sampling and conversion timing . . . 382
Table 57. Interrupt event summary . . . 401
Table 58. CAN1 register mapping . . . 411
Table 59. CAN2 register mapping . . . 412
Table 60. C-CAN register memory space summary . . . 423
Table 61. IF1 and IF2 message interface register sets . . . 429
Table 62. Parameters of the CAN bit time . . . 453
Table 63. Reset event definition . . . 472
Table 64. Reset events summary . . . 495
Table 65. PORT0 latched configuration for the different reset events . . . 497
Table 66. Power reduction modes summary . . . 503
Table 67. Output pin state during Idle and power down modes . . . 512
Table 68. General purpose registers (GPRs) . . . 516
Table 69. General purpose registers (GPRs) bit wise addressing . . . 516
Table 70. Special function registers ordered by name . . . 517
Table 71. Special function registers ordered by address . . . 524
Table 72. X-Registers ordered by name . . . 531
Table 73. X-Registers ordered by address . . . 536
Table 74. Flash registers ordered by name . . . 541
Table 75. Flash registers ordered by address . . . 541
Table 76. Stack size selection . . . 548
Table 77. Document revision history . . . 561

List of figures

Figure 1. ST10F276x functional block diagram . . . 23
Figure 2. CPU block diagram . . . 24
Figure 3. Clock block diagram . . . 30
Figure 4. ST10F276 memory mapping (user mode: Flash Read operation) . . . 41
Figure 5. Storage of words, bytes and bits in a byte organized memory . . . 42
Figure 6. On-chip RAM and SFR/ESFR areas . . . 45
Figure 7. Location of the PEC pointers . . . 48
Figure 8. CPU block diagram . . . 54
Figure 9. Sequential instruction pipelining . . . 55
Figure 10. Standard branch instruction pipelining . . . 56
Figure 11. Cache jump instruction pipelining . . . 56
Figure 12. Addressing via the code segment pointer . . . 71
Figure 13. Addressing via the data page pointers . . . 73
Figure 14. Register bank selection via register CP . . . 75
Figure 15. Implicit CP use by short GPR addressing modes . . . 75
Figure 16. MAC architecture . . . 83
Figure 17. Example of parallel data move . . . 86
Figure 18. Pipeline diagram for MAC interrupt response time . . . 90
Figure 19. Priority levels and PEC channels . . . 102
Figure 20. Mapping of PEC pointers into the IRAM . . . 106
Figure 21. Task status saved on the system stack . . . 109
Figure 22. Pipeline diagram for interrupt response time . . . 110
Figure 23. Pipeline diagram for PEC response time . . . 112
Figure 24. X-Interrupt basic structure . . . 118
Figure 25. SFRs, XBUS registers and pins associated with the parallel ports . . . 136
Figure 26. Output drivers in push-pull mode and in open drain mode . . . 137
Figure 27. Hysteresis concept . . . 138
Figure 28. PORT0 I/O and alternate functions . . . 141
Figure 29. Block diagram of a PORT0 pin . . . 142
Figure 30. PORT1 I/O and alternate functions . . . 144
Figure 31. Block diagram of input section of a P1L pin . . . 145
Figure 32. Block diagram of a PORT1 pin . . . 146
Figure 33. Port2 I/O and alternate functions . . . 149
Figure 34. Block diagram of a Port2 pin . . . 150
Figure 35. Port3 I/O and alternate functions . . . 152
Figure 36. Block diagram of a Port3 pin . . . 153
Figure 37. Block diagram of P3.15 (CLKOUT) and P3.12 (BHE/WRH) pins . . . 154
Figure 38. Port4 I/O and alternate functions . . . 156
Figure 39. Block diagram of Port4 pins 3...0 . . . 157
Figure 40. Block diagram of P4.4 pin . . . 158
Figure 41. Block diagram of P4.5 pin . . . 159
Figure 42. Block diagram of P4.6 pin . . . 160
Figure 43. Block diagram of P4.7 pin . . . 161
Figure 44. Port5 I/O and alternate functions . . . 163
Figure 45. Block diagram of a Port5 pin . . . 163
Figure 46. Port6 I/O and alternate functions . . . 166
Figure 47. Block diagram of Port6 pins 4...0 . . . 167
Figure 48. Block diagram of P6.5 pin . . . 168

Figure 49. Port7 I/O and alternate functions . . . 170
Figure 50. Block diagram of Port7 pins 3...0 . . . 171
Figure 51. Block diagram of Port7 pins 7...4 . . . 172
Figure 52. Port8 I/O and alternate functions . . . 175
Figure 53. Block diagram of Port8 pins 3...0 . . . 176
Figure 54. Block diagram of P8.4 and P8.5 pins . . . 177
Figure 55. Block diagram of P8.6 pin . . . 178
Figure 56. RPD external RC circuit . . . 180
Figure 57. SFRs and port pins associated with the external bus interface . . . 182
Figure 58. Multiplexed bus cycle . . . 184
Figure 59. De-multiplexed bus cycle . . . 185
Figure 60. Switching from de-multiplexed to multiplexed bus mode . . . 187
Figure 61. Programmable external bus cycle . . . 190
Figure 62. ALE length control . . . 191
Figure 63. Memory cycle time . . . 192
Figure 64. Memory tri-state time . . . 193
Figure 65. Read / write delay . . . 194
Figure 66. READY/READY controlled bus cycles . . . 195
Figure 67. Chip select delay . . . 196
Figure 68. Address window arbitration . . . 202
Figure 69. Sharing external resources using slave mode . . . 205
Figure 70. External bus arbitration, releasing the bus . . . 206
Figure 71. External bus arbitration, (regaining the bus) . . . 207
Figure 72. Memory mapping (User mode: Flash read operations / XADRS = 800Bh) . . . 210
Figure 73. Memory mapping (User mode: Flash read operations / XADRS = C00Ah) . . . 211
Figure 74. EA / VSTBY external circuit . . . 214
Figure 75. SFRs and port pins associated with timer block GPT1 . . . 216
Figure 76. GPT1 block diagram . . . 217
Figure 77. Core timer T3 in timer mode . . . 219
Figure 78. Core timer T3 in gated timer mode . . . 221
Figure 79. Core timer T3 in counter mode . . . 221
Figure 80. Core timer T3 in incremental interface mode . . . 222
Figure 81. Connection of the encoder to the ST10F276 . . . 223
Figure 82. Evaluation of the incremental encoder signals . . . 224
Figure 83. Evaluation of the incremental encoder signals . . . 224
Figure 84. Auxiliary timer in counter mode . . . 226
Figure 85. Concatenation of core timer T3 and an auxiliary timer . . . 228
Figure 86. GPT1 auxiliary timer in reload mode . . . 228
Figure 87. GPT1 timer reload configuration for PWM generation . . . 230
Figure 88. GPT1 auxiliary timer in capture mode . . . 231
Figure 89. SFRs and port pins associated with timer block GPT2 . . . 233
Figure 90. GPT2 block diagram . . . 234
Figure 91. Block diagram of core timer T6 in timer mode . . . 236
Figure 92. Block diagram of core timer T6 in gated timer mode . . . 237
Figure 93. Block diagram of core timer T6 in counter mode . . . 238
Figure 94. Block diagram of auxiliary timer T5 in counter mode . . . 240
Figure 95. Concatenation of core timer T6 and auxiliary timer T5 . . . 242
Figure 96. GPT2 register CAPREL in capture mode . . . 242
Figure 97. GPT2 register CAPREL in reload mode . . . 243
Figure 98. GPT2 register CAPREL in capture-and-reload mode . . . 244
Figure 99. SFRs and port pins associated with ASC0 . . . 246
Figure 100. Asynchronous mode of serial channel ASC0 . . . 249

Figure 101. Asynchronous 8-bit data frames . . . 250
Figure 102. Asynchronous 9-bit data frames . . . 251
Figure 103. Synchronous mode of serial channel ASC0 . . . 252
Figure 104. ASC0 interrupt generation . . . 256
Figure 105. XBUS registers and port pins associated with XASC . . . 258
Figure 106. Asynchronous mode of serial channel XASC . . . 262
Figure 107. Asynchronous 8-bit data frames . . . 262
Figure 108. Asynchronous 9-bit data frames . . . 264
Figure 109. Synchronous mode of serial channel XASC . . . 265
Figure 110. XASC interrupt generation . . . 269
Figure 111. SFRs and port pins associated with the SSC . . . 271
Figure 112. Synchronous serial channel SSC block diagram . . . 272
Figure 113. Serial clock phase and polarity options . . . 276
Figure 114. SSC full duplex configuration . . . 276
Figure 115. SSC half duplex configuration . . . 279
Figure 116. SSC error interrupt control . . . 282
Figure 117. XBUS registers and port pins associated with the XSSC . . . 285
Figure 118. Synchronous serial channel XSSC block diagram . . . 286
Figure 119. Serial clock phase and polarity options . . . 290
Figure 120. XSSC full duplex configuration . . . 291
Figure 121. XSSC half duplex configuration . . . 293
Figure 122. SFRs and port pins associated with the watchdog timer . . . 297
Figure 123. Watchdog timer block diagram . . . 297
Figure 124. ST10F276x new standard bootstrap loader program flow . . . 304
Figure 125. Booting steps for ST10F276 . . . 306
Figure 126. Hardware provisions to activate the BSL . . . 307
Figure 127. Memory configuration after reset . . . 308
Figure 128. UART bootstrap loader sequence . . . 310
Figure 129. Baudrate deviation between host and ST10F276 . . . 312
Figure 130. CAN bootstrap loader sequence . . . 313
Figure 131. Bitrate measurement over a predefined zero-frame . . . 316
Figure 132. Reset boot sequence . . . 325
Figure 133. SFRs and Port Pins associated with the CAPCOM units . . . 327
Figure 134. CAPCOM unit block diagram . . . 329
Figure 135. Block diagram of CAPCOM timers T0 and T7 . . . 330
Figure 136. Block diagram of CAPCOM timers T1 and T8 . . . 330
Figure 137. Capture mode block diagram . . . 336
Figure 138. Compare mode 0 and 1 block diagram . . . 338
Figure 139. Timing example for compare modes 0 and 1 . . . 339
Figure 140. Compare mode 2 and 3 block diagram . . . 340
Figure 141. Timing example for compare modes 2 and 3 . . . 340
Figure 142. Double register compare mode block diagram . . . 342
Figure 143. Timing example for double register compare mode . . . 343
Figure 144. SFRs and port pins associated with the PWM module . . . 346
Figure 145. PWM channel block diagram . . . 347
Figure 146. Operation and output waveform in mode 0 . . . 348
Figure 147. Operation and output waveform in mode 1 . . . 349
Figure 148. Operation and output waveform in burst mode . . . 350
Figure 149. Operation and output waveform in single shot mode . . . 351
Figure 150. PWM output signal generation . . . 356
Figure 151. XBUS registers and port pins associated with the XPWM module . . . 358
Figure 152. XPWM channel block diagram . . . 359

Figure 153. Operation and output waveform in mode 0 . . . 360
Figure 154. Operation and output waveform in mode 1 . . . 361
Figure 155. Operation and output waveform in burst mode . . . 362
Figure 156. Operation and output waveform in single shot mode . . . 363
Figure 157. XPWM output signal generation . . . 369
Figure 158. SFRs, XBUS registers and port pins associated with the A/D converter . . . 371
Figure 159. Analog / digital converter block diagram . . . 372
Figure 160. Auto scan conversion mode example . . . 376
Figure 161. Wait for read mode example . . . 377
Figure 162. Channel injection example . . . 379
Figure 163. Channel injection example with wait for read . . . 380
Figure 164. A/D conversion characteristic . . . 385
Figure 165. A/D converter input pins scheme . . . 386
Figure 166. Charge sharing timing diagram during sampling phase . . . 387
Figure 167. Anti-aliasing filter and conversion rate . . . 389
Figure 168. I2C bus protocol . . . 393
Figure 169. I^2C interface block diagram . . . 395
Figure 170. Transfer sequencing . . . 399
Figure 171. Event flags and interrupt generation . . . 401
Figure 172. Connection to single CAN bus via separate CAN transceivers . . . 415
Figure 173. Connection to single CAN bus via one common transceiver . . . 415
Figure 174. Connection to two different CAN buses (e.g. for gateway application) . . . 416
Figure 175. Connection to one CAN bus with internal parallel mode enabled . . . 416
Figure 176. Block diagram of the C-CAN . . . 418
Figure 177. CAN core in silent mode . . . 420
Figure 178. CAN core in loop back mode . . . 421
Figure 179. CAN core in loop back combined with silent mode . . . 421
Figure 180. Data transfer between IFx registers and message RAM . . . 446
Figure 181. CPU handling of a FIFO buffer . . . 451
Figure 182. Bit timing . . . 453
Figure 183. The propagation time segment . . . 454
Figure 184. Synchronization on “late” and “early” edges . . . 456
Figure 185. Filtering of short dominant spikes . . . 457
Figure 186. Structure of the CAN Core’s CAN protocol controller . . . 461
Figure 187. SFRs associated with the RTC . . . 465
Figure 188. XBUS registers associated with the RTC . . . 465
Figure 189. RTC block diagram . . . 466
Figure 190. Prescaler register . . . 468
Figure 191. Divider counters . . . 469
Figure 192. Asynchronous power-on RESET (EA = 1) . . . 474
Figure 193. Asynchronous power-on RESET (EA = 0) . . . 475
Figure 194. Asynchronous hardware RESET (EA = 1) . . . 476
Figure 195. Asynchronous hardware RESET (EA = 0) . . . 477
Figure 196. Synchronous short / long hardware RESET (EA = 1) . . . 480
Figure 197. Synchronous short / long hardware RESET (EA = 0) . . . 481
Figure 198. Synchronous long hardware RESET (EA = 1) . . . 482
Figure 199. Synchronous long hardware RESET (EA = 0) . . . 483
Figure 200. SW / WDT unidirectional RESET (EA = 1) . . . 485
Figure 201. SW / WDT unidirectional RESET (EA = 0) . . . 486
Figure 202. SW / WDT bidirectional RESET(EA = 1) . . . 488
Figure 203. SW / WDT bidirectional RESET (EA = 0) . . . 489
Figure 204. SW / WDT bidirectional RESET (EA = 0) followed by a HW RESET . . . 490

Figure 205. Minimum external reset circuitry . . . 491
Figure 206. System reset circuit . . . 492
Figure 207. Example of software or watchdog bidirectional reset (EA = 1) . . . 493
Figure 208. Example of software or watchdog bidirectional reset (EA = 0) . . . 494
Figure 209. PORT0 bits latched into the different registers after reset . . . 497
Figure 210. Transitions between Idle mode and active mode . . . 504
Figure 211. RPD pin: external circuit to exit power down . . . 507
Figure 212. Simplified power down exit circuitry . . . 508
Figure 213. Power down exit sequence using an external interrupt (PLL x 2) real time clock and power down mode . . . 509
Figure 214. Physical stack address generation . . . 549
Figure 215. Local registers . . . 553

1 Architectural overview

ST10F276 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 ST10F276 (see *Figure 1*).

1.1 Basic CPU concepts and optimization

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. ST10F276x functional block diagram

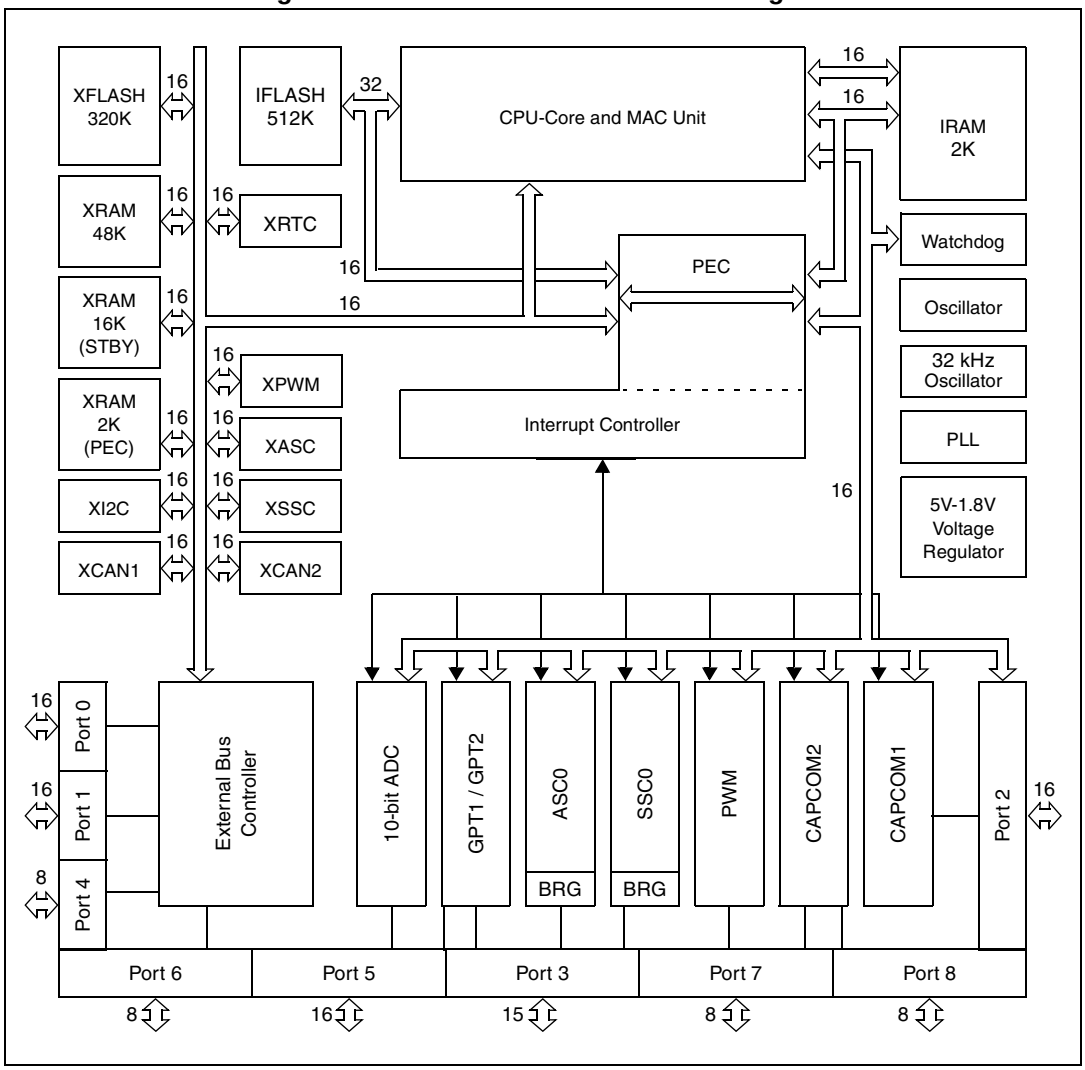
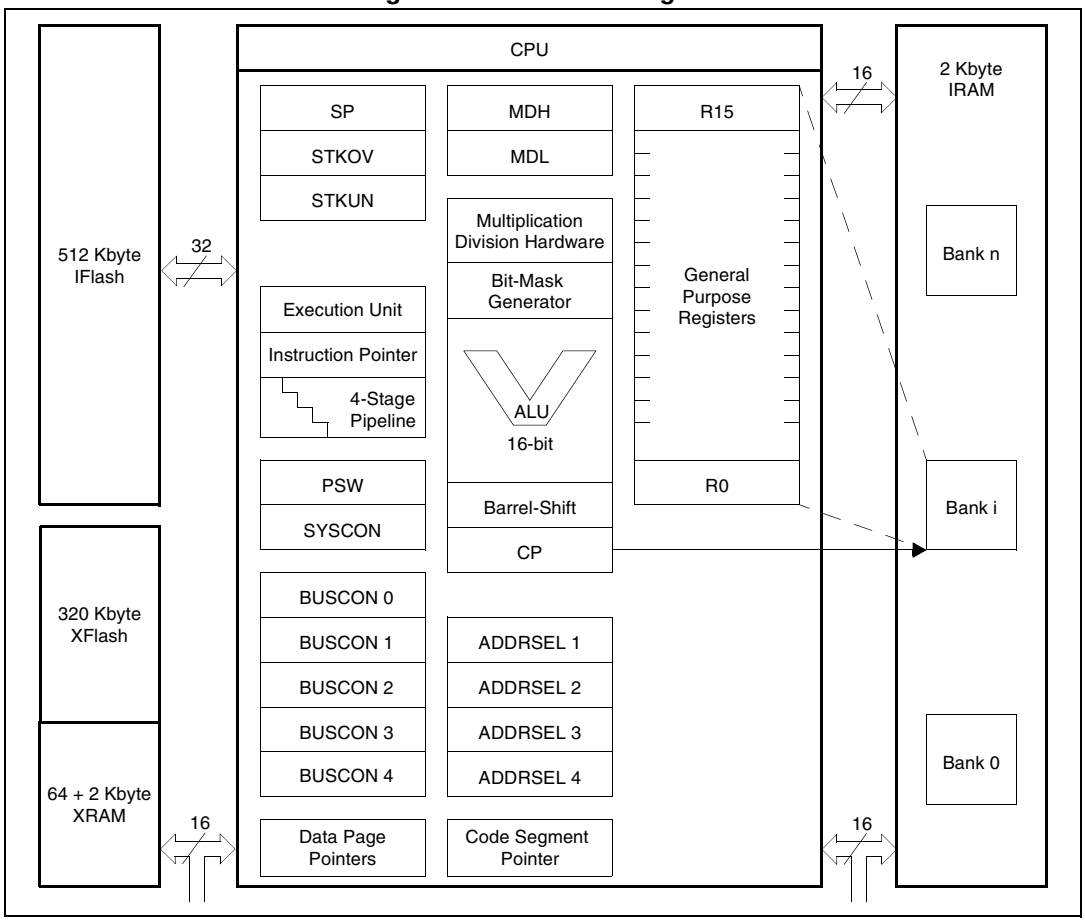


Figure 2. CPU block diagram



1.1.1 High instruction bandwidth / fast execution

Most of the ST10F276's instructions are executed in one instruction cycle. For example, shift and rotate instructions are processed independent of the number of bits to be shifted. Multiple-cycle instructions have been optimized: branches are carried out in 2 instruction cycles, 16 × 16 bit multiplication in 5 instruction cycles and a 32/16-bit division in 10 instruction cycles. The jump cache reduces the execution time of repeatedly performed jumps in a loop, from 2 instruction cycles to 1 instruction cycle.

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

- Fetch: In this stage, an instruction is fetched from the internal Flash or RAM or from the external memory, based on the current IP value.
- Decode: In this stage, the previously fetched instruction is decoded and the required operands are fetched.
- Execute: In this stage, the specified operation is performed on the previously fetched operands.
- Write back: In this stage, the result is written to the specified location.

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

1.1.2 High function 8-bit and 16-bit ALU

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

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

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

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

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

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

A set of consistent flags is automatically updated in the PSW register 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.

1.1.3 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.

1.1.4 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.

1.1.5 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 ST10F276 instruction set. Possible operand types are bits, bytes and words. Specific instruction support the conversion (extension) of bytes to words. A variety of direct, indirect or immediate addressing modes are provided to

specify the required operands.

1.1.6 Programmable multiple priority interrupt system

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

- Peripheral Event Controller (PEC): This processor is used to off-load many interrupt requests from the CPU. It avoids the overhead of entering and exiting interrupt or trap routines by performing single cycle interrupt-driven byte or word data transfers between any two locations in segment 0 with an optional increment of either the PEC source or the destination pointer. Just one cycle is 'stolen' from the current CPU activity to perform a PEC service.
- Multiple Priority Interrupt Controller: This controller allows all interrupts to be placed at any specified priority. Interrupts may also be grouped, which provides the user with the ability to prevent similar priority tasks from interrupting each other. For each of the possible interrupt sources there is a separate control register, which contains an interrupt request flag, an interrupt enable flag and an interrupt priority bit-field. Once having been accepted by the CPU, an interrupt service can only be interrupted by a higher prioritized service request. For standard interrupt processing, each of the possible interrupt sources has a dedicated vector location.
- Multiple Register Banks: This feature allows the user to specify up to sixteen general purpose registers located anywhere in the IRAM. 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 5 to 12 CPU clock periods, the ST10F276 is capable of fast reaction to non-deterministic events.

The ST10F276 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.

1.2 On-chip system resources

The ST10F276 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.

1.2.1 Peripheral event control and interrupt control

The Peripheral Event Controller (PEC) 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 ST10F276 has eight PEC channels each of which offers such fast interrupt-driven data transfer capabilities.

1.2.2 Memory areas

The memory space of the ST10F276 is organized as a unified memory which means that code memory, data memory, registers and I/O ports are organized within the same linear address space which covers up to 16 Mbytes. The entire memory space can be accessed byte wise or word wise. Particular portions of the on-chip memory have additionally been made directly bit addressable.

A 2 Kbyte 16-bit wide IRAM provides fast access to General Purpose Registers (GPRs), user data (variables) and system stack. The IRAM 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 IRAM 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 IRAM 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 IRAM 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 66 Kbyte 16-bit wide on-chip XRAM provides fast access to user data (variables), user stacks and code. The on-chip XRAM is an X-Peripheral and appears to the software as an

external RAM. Therefore it cannot store register banks and is not bit addressable. The XRAM allows 16-bit accesses with maximum speed. A portion of the on-chip XRAM (16 Kbytes) represents the Stand-by RAM, which can be maintained biased through $\overline{EA}/V_{STBY}$ pin when main supply V_{DD} is turned off.

A 512 Kbyte on-chip internal Flash (IFlash) 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 IFlash is the fastest of all possible alternatives.

A 320 Kbyte on-chip extended Flash (XFlash) is also provided for both code and constant data storage. It is seen as an X-Peripheral and appears to the software as an external Flash: it allows 16-bit accesses with maximum speed up to 40 MHz. A wait state should be added when a higher CPU frequency is used.

For Special Function Registers 1024 bytes of the address space are reserved. The standard Special Function Register area (SFR) uses 512 bytes, while the Extended Special Function Register area (ESFR) uses the other 512 bytes. (E)SFRs are word wide registers which are used for controlling and monitoring functions of the different on-chip units. Unused ESFR addresses are reserved for future members of the ST10F276 family.

1.2.3 External bus interface

In addition to the internal memory, the application can address up to 16 Mbytes of external memory via the external bus interface.

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

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

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

The de-multiplexed bus modes use PORT1 for addresses and PORT0 for data input/output. The multiplexed bus modes use PORT0 for both addresses and data input/output.

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

To address more than 64 Kbytes of external memory, Port4 is used to generate the address lines A16...A23. Otherwise Port4 can be used as standard I/O.

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 XFlash, the on-chip CAN-Modules, the XASC, the XSSC, the XPWM, the I^2C interface, the RTC are all examples for these X-Peripherals.

1.3 Clock generator

The on-chip clock generator provides the ST10F276 with its basic clock signal that controls the activities of the controller hardware. Its oscillator amplifier can run with an external crystal and appropriate oscillator circuitry (see *Section 7: Dedicated pins on page 179*), or can be driven by an external clock source.

Direct Drive mode allows to feed the device with an external clock signal to provide directly the clock to the CPU, up to maximum internally allowed speed. In this mode, the on-chip oscillator amplifier is bypassed, so there is no limit imposed by the bandwidth of the amplifier circuit itself.

On the contrary, for all the other configurations, the on-chip oscillator amplifier is not bypassed, so the external clock can be provided by a crystal or resonator only, according to the limited frequency ranges (refer to datasheet for more details).

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.

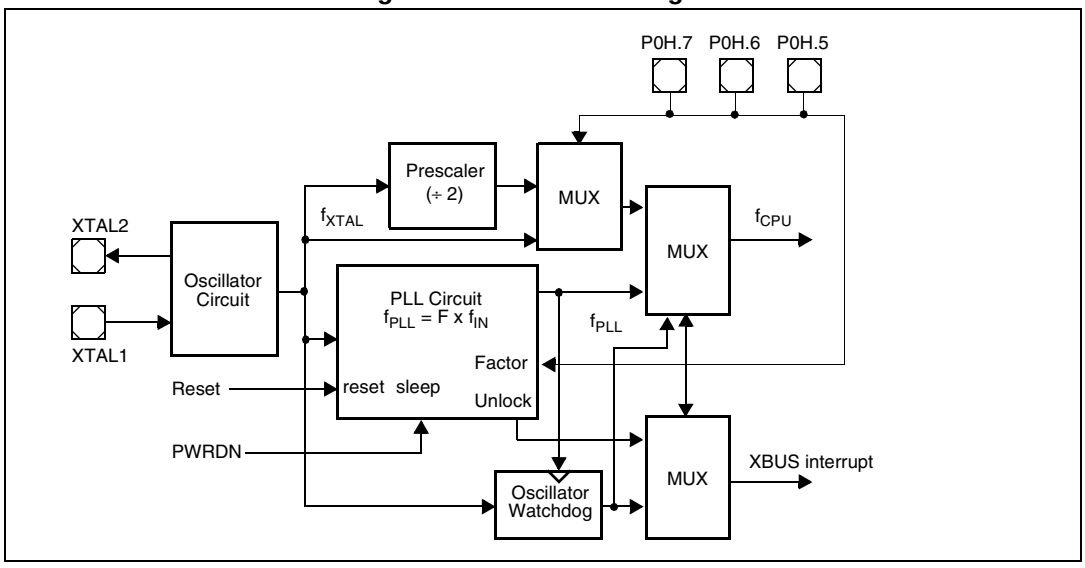
While the CPU clock is stopped during idle mode, the peripheral clock keeps running. Both clocks are switched off when the Power Down mode is entered.

The on-chip PLL circuit allows operation of the ST10F276 with a low frequency external clock while still providing maximum performance.

The PLL multiplies the external clock frequency by a selectable factor F (0.5, 1, 3, 4, 5, 8, 10, 16) 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 bypassed (see *Section 1.3.4: Oscillator watchdog (OWD) on page 32*).

Figure 3. Clock block diagram



1.3.1 PLL operation

The PLL is enabled except when P0H.[7..5] = ‘011’ or ‘001’ during reset (Direct Drive and Prescaler modes). At Power-On, the PLL provides a stable clock signal in less than 1ms after V_{DD} has reached 5V±10%, even if there is no external clock signal (in this case, the PLL will run on its basic frequency of 750 kHz to 3 MHz). Refer to datasheet for more details about PLL characteristics.

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

Note: The ST10F276 is required to operate on the desired CPU clock directly after reset: make sure that RSTIN remains active until the PLL has locked, on the contrary unlock detection circuit immediately after reset will disconnect crystal reference clock path from the PLL input, so the CPU clock will be provided by the PLL free running frequency.

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 (few CPU clock cycles), it remains below the specified value (refer to datasheet for details).

When the PLL is detected no longer locked (no longer stable), it generates an interrupt request (on the PLL Unlock 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 (750 kHz to 3 MHz). The basic frequency is still generated and allows the CPU to execute emergency actions in case of an external clock loss.

1.3.2 Prescaler operation

When pins P0H.[7..5] = ‘001’ during reset, the CPU clock is derived from the internal oscillator (input clock signal) by a 2:1 prescaler. Note that it is not possible to force a clock signal through an external clock generator unless Direct Drive is selected.

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

The PLL is still running on its basic frequency of 750 kHz to 3 MHz, and delivers the clock signal for the Oscillator Watchdog, except if bit OWDDIS is set: in this case the PLL is switched off.

1.3.3 Direct drive

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

The PLL runs on its basic frequency of 750 kHz to 3 MHz, and delivers the clock signal for the Oscillator Watchdog, except if bit OWDDIS is set: in this case the PLL is switched off.

1.3.4 Oscillator watchdog (OWD)

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

The oscillator watchdog operates as follows:

- The Oscillator WatchDog (OWD) is enabled by default after reset. To disable the OWD, set bit OWDDIS of the SYSCON register.
- When the OWD is enabled, the PLL runs on its free-running frequency, and increments the Oscillator Watchdog counter.
- On each transition of XTAL1 pin, the Oscillator Watchdog counter 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 750 kHz to 3 MHz), and the Oscillator Watchdog Interrupt Request is flagged.

The CPU clock will not switch back to the external clock even if a valid external clock is applied 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.

1.4 On-chip peripheral blocks

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

The built in peripherals are used for interfacing the CPU to the external world, or to provide on-chip functions. The ST10F276 generic peripherals are:

- Nine I/O ports with a total of 111 I/O lines,
- Two Serial Interfaces (ASC0 and SSC),
- Two General Purpose Timer Blocks (GPT1 and GPT2),
- A Watchdog Timer,
- Two 16-channel Capture / Compare units (CAPCOM1 and CAPCOM2),
- A 4-channel Pulse Width Modulation unit (PWM),
- A 10-bit Analog / Digital Converter.

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.

In order to enhance the performance of the device, a set of additional on-chip X-Peripherals are available on ST10F276 and controlled through dedicated set of registers:

- Two CAN interfaces,
- Two additional Serial Interfaces (XASC and XSSC),
- A I^2C Serial Interface,
- An additional 4-channel Pulse Width Modulation unit (XPWM),
- A Real Time Clock module.

1.4.1 Peripheral interfaces

The on-chip peripherals generally have two different types of interfaces: an interface to the CPU and an interface to external hardware. Communication between 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 such as end of task, new event, or errors.

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.

Similarly, the on-chip X-Peripherals communicate with the CPU through a dedicated set of registers and dedicated structure of interrupt management system.

1.4.2 Peripheral timing

Internal operation of CPU and peripherals is based on the CPU clock (f_{CPU}). The on-chip oscillator derives the CPU clock from the crystal or from the external clock signal.

The clock signal which is gated to the peripherals is independent from the clock signal which feeds the CPU. During Idle mode the CPU’s clock is stopped while the peripherals continue their operation.

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.

1.4.3 Programming hints

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

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

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

Reserved bit: some of the bits which are contained in the ST10F276's SFRs are marked as 'Reserved'. User software must write '0's to reserved bits.

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

1.4.4 Parallel ports

The ST10F276 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. Four CAPCOM2 input-only lines and additional ADC channels are also mapped on this port.
- Port2, accepts the fast external interrupt inputs and provides inputs/outputs for CAPCOM1 unit.
- Port3 includes the alternate functions of timers, serial interfaces, the optional bus control signal $\overline{BHE}$ and the system clock output (CLKOUT).
- Port4 outputs the additional segment address bit A16 to A23 in systems where segmentation is enabled to access more than 64 Kbytes of memory. CAN modules and I^2C serial interface are alternate function on this port also.
- Port5 is used as analog input channels of the A/D converter or as timer control signals.
- Port6 provides optional bus arbitration signals ($\overline{BREQ}$, $\overline{HLDA}$, $\overline{HOLD}$), chip select signals and XSSC signals.
- Port7 provides the output signals from the PWM unit and inputs/outputs for the CAPCOM2 unit.
- Port8 provides inputs/outputs for the CAPCOM2 unit, for the XPWM and for the XASC.

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

1.4.5 Serial channels

Serial communication with other microcontrollers, processors, terminals or external peripheral components is provided by four serial interfaces with different functionality: two

Asynchronous/Synchronous Serial Channels (ASC0 and XASC) and two High-Speed Synchronous Serial Channels (SSC and XSSC).

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 three separate interrupt vectors are provided on channel SSC, four vectors are provided on channel ASC0.

In asynchronous mode, 8- or 9-bit data frames are transmitted or received, preceded by a start bit and terminated by one or two stop bits. For multiprocessor communication, a mechanism to distinguish address from data byte has been included (8-bit data plus wake up bit mode).

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

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

The XASC is another USART which is functionally identical with the ASC0. The XASC is an X-Peripheral (no bit handling) and supports three interrupt vectors. The port line and interrupt handling is slightly different from that of the ASC0.

Similarly, the XSSC is a Synchronous Serial link functionally identical with the SSC. The XSSC is an X-Peripheral (no bit handling) and supports three interrupt sources. The port line and interrupt handling is slightly different from that of the SSC.

1.4.6 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, pulse multiplication or incremental interface.

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.

1.4.7 Watchdog timer

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

The Watchdog Timer is always enabled after a reset of the chip, and can only be disabled in the time interval until the EINIT (end of initialization) instruction has been executed. In this way the chip's start-up procedure is always monitored. The software must be designed to service the Watchdog Timer before it overflows. If, due to hardware or software related failures, the software fails to do so, the Watchdog Timer overflows and generates an internal hardware reset and pulls the $\overline{\text{RSTOUT}}$ pin low in order to allow external hardware components to be reset.

The Watchdog Timer is a 16-bit timer, clocked with the system clock divided either by 2 or by 128. The high byte of the watchdog Timer register can be set to a pre-specified reload value (stored in WDTREL) in order to allow further variation of the monitored time interval. Each time it is serviced by the application software, the high byte of the Watchdog Timer is reloaded.

1.4.8 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 (PWM), 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.

1.4.9 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).

The XPWM is an additional 4-channel PWM unit, functionally identical with the standard PWM. The XPWM is an X-Peripheral (no bit handling) and support 1 interrupt source. The port line and interrupt handling is slightly different from that of the standard PWM unit.

1.4.10 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. An additional eight multiplexed channels are also available on Port1 with a reduced accuracy.

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 24 analog input channels, the remaining channel inputs can be used as digital input/output port pins (note that Port5 is input only, while Port1 is input/output).

The A/D converter of the ST10F276 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 sampled and converted in sequence.
- 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.

Also, in the Wait for ADDAT read mode, a conversion will not be started as long as the result from the previous one has not been read.

1.4.11 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 A and 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 32 message objects: each message object has its own identifier mask. Message objects may be concatenated in a Programmable FIFO mode.

All message objects can be updated independent from the other objects and are equipped for the maximum message length of 8 bytes. The bit timing is derived from the X-Peripheral clock (XCLK, typically equal to CPU clock, apart from idle mode) and is programmable up to a data rate of 1M Baud. The CAN Module uses two pins to interface to a bus transceiver.

1.4.12 I^2C serial interface

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

The I^2C module uses two pins to interface with the external serial bus. Data can be transferred at speeds up to 400 Kbit/s. The I^2C is an X-Peripheral (no bit handling) and supports three interrupt sources.

Note: Once I^2C module is enabled, the port pins associated with the peripheral feature open drain drivers only, as required by the standard bus specification.

1.5 Real time clock

The Real Time Clock is an independent timer, which clock is directly derived from the oscillator clock (either the main on-chip oscillator or the 32 kHz on-chip oscillator), so that it can be maintained running even in Power Down mode (if enabled to) or in Stand-by mode (only if 32 kHz oscillator is used). Registers access is implemented onto the XBUS. This module is designed for the following purposes:

- Generate the current time and date for the system.
- Provide cyclic time based interrupt on Port2 external interrupts every ‘RTC basic clock tick’ and every *n* ‘RTC basic clock tick’ (*n* is programmable) if enabled.
- Long term measurements (thanks to a 58-bit timer).
- Exit the ST10F276 from Power Down mode (if PWDCFG of SYSCON set) after a programmed delay.

1.6 Protected bits

The ST10F276x MCU provides up to 106 protected bits. These bits are modified by the on-chip hardware during special events such as Power-On reset, power failure, or application hardware. These bits cannot be modified by some wrong software accesses.

Table 1. 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 flags
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 bits of Port2
P7	P7.7...P7.0	All bits of Port7
P8	P8.7...P8.0	All bits of Port8
XPyIC (y=3...0)	XPyIR (y=3...0)	X-Peripheral y interrupt request flag

Note: Σ = 106 protected bit.

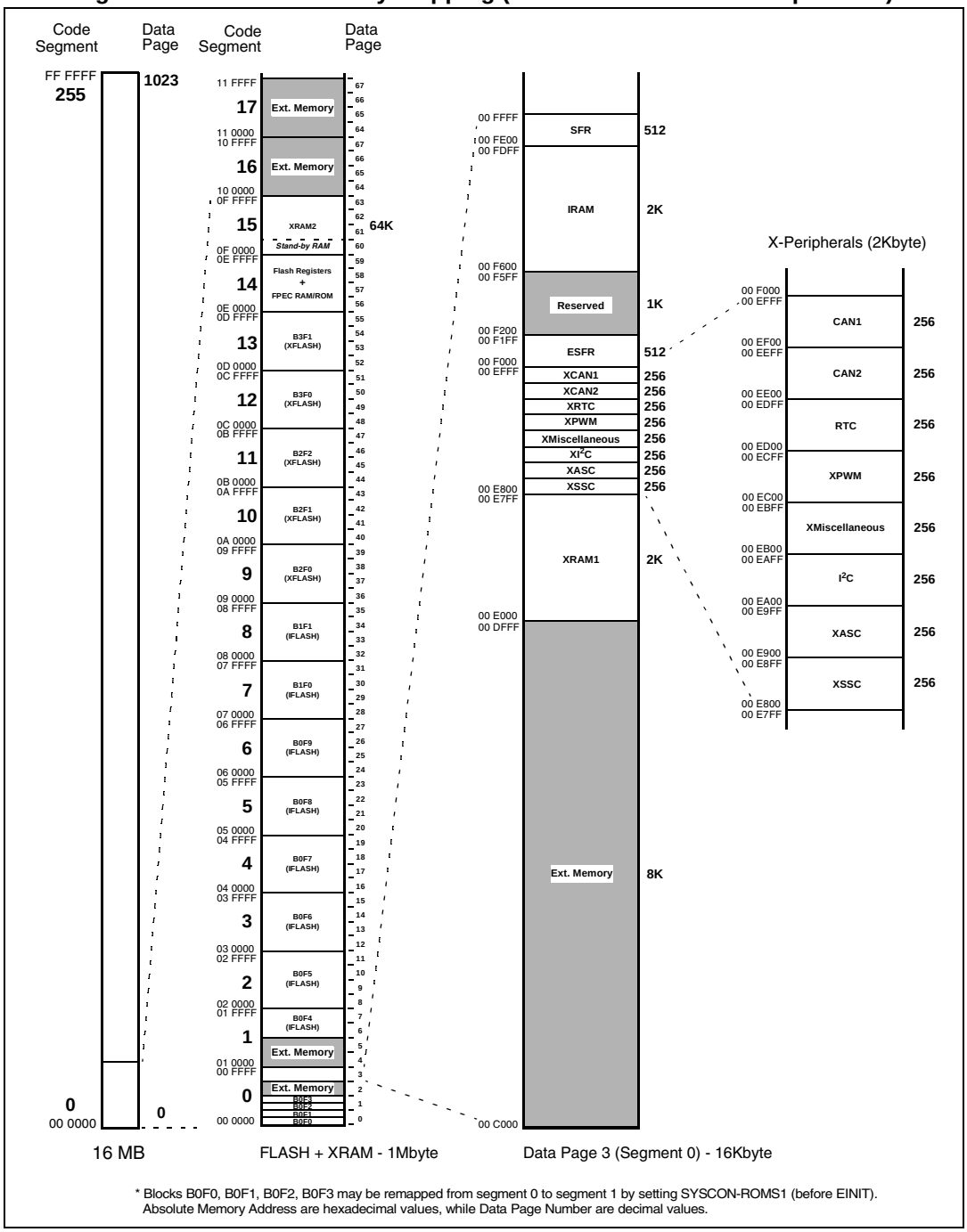
2 Memory organization

The memory space of the ST10F276 is organized as a unified memory. Code memory, data memory, registers and I/O ports are organized within the same linear address space.

All of the physically separated memory areas, including on-chip IFlash, IRAM, the internal Special Function Register Areas (SFRs and ESFRs), the address areas for integrated XBUS peripherals and external memory are mapped into one common address space.

The ST10F276 provides a total addressable memory space of 16 Mbytes. This address space is arranged as 256 segments of 64 Kbytes each, and each segment is again subdivided into four data pages of 16 Kbytes each (see *Figure 4 on page 41*).

Figure 4. ST10F276 memory mapping (user mode: Flash Read operation)



* Blocks B0F0, B0F1, B0F2, B0F3 may be remapped from segment 0 to segment 1 by setting SYSCON-ROMS1 (before EINIT).
Absolute Memory Address are hexadecimal values, while Data Page Number are decimal values.

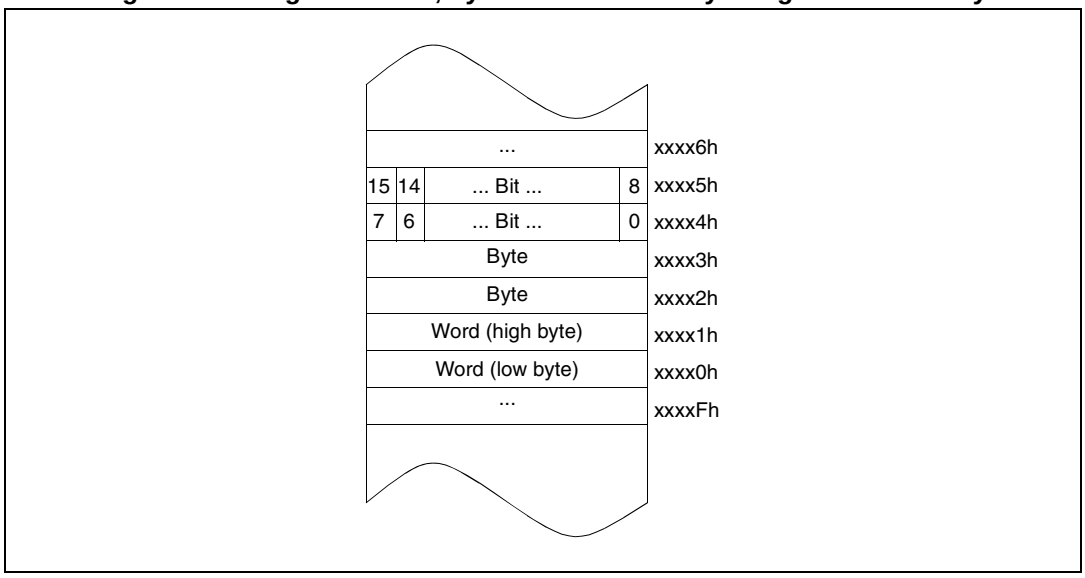
2.1 Word, byte and bit storage

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

Double words (code only) are stored in ascending memory locations as two subsequent words. Single bit are always stored in the specified bit position at a word address.

Bit position 0 is the least significant bit of the byte at an even byte address, and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported for a part of the Special Function Registers, a part of the IRAM and for the General Purpose Registers.

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



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

2.2 On-chip Flash

The ST10F276 reserves an address area of 512+320 Kbytes for on-chip Flash memory.

In standard mode (the normal operating mode) the first portion (512 Kbytes) of the Flash (IFlash) appears like an on-chip ROM with the same timing and functionality. The IFlash module offers a fast access time, allowing zero wait-state access with CPU frequency up to 64 MHz. Instruction fetches and data operand reads are performed with all addressing modes of the ST10F276 instruction set. The second portion (320 Kbytes) appears like an on-chip X-Peripheral: it can be accessed like an external memory in 16-bit demultiplexed bus-mode without read/write delay at zero wait-states with CPU frequency up to 40 MHz, while one wait-state should be added for higher frequency rates.

In order to optimize the programming time of the internal Flash, up to four different Banks divided in blocks of 8 Kbytes, 16 Kbytes, 32 Kbytes and 64 Kbytes can be used. Reading from one Bank while programming another Bank is possible. For a complete memory mapping summary, see details in *Table 3* and *Table 4*.

In the table below, the Segment 8 address range mapping is summarized, varying ROMEN (related with $\overline{EA}$ pin status under reset) and XPEN bits status in SYSCON register and XPERCON (bits XRAM2EN and XFLASHEN) programming.

Table 2. Segment 8 address range mapping

ROMEN	XPEN	XRAM2EN	XFLASHEN	Segment 8
0	0	x	x	External Memory
0	1	0	0	External Memory
0	1	1	x	Reserved
0	1	x	1	Reserved
1	x	x	x	IFlash (B1F1)

Note: The symbol 'x' in the table above stands for 'do not care'.

Table 3. 512 Kbyte IFlash memory block organization

Block	Addresses (ROMS1 = '0')	Addresses (ROMS1 = '1')	Size (bytes)
B0F0	00'0000h - 00'1FFFh	01'0000h - 00'1FFFh	8K
B0F1	00'2000h - 00'3FFFh	01'2000h - 00'3FFFh	8K
B0F2	00'4000h - 00'5FFFh	01'4000h - 00'5FFFh	8K
B0F3	00'6000h - 00'7FFFh	01'6000h - 00'7FFFh	8K
B0F4	01'8000h - 01'FFFFh	01'8000h - 01'FFFFh	32K
B0F5	02'0000h - 02'FFFFh	02'0000h - 02'FFFFh	64K
B0F6	03'0000h - 03'FFFFh	03'0000h - 03'FFFFh	64K
B0F7	04'0000h - 04'FFFFh	04'0000h - 04'FFFFh	64K
B0F8	05'0000h - 05'FFFFh	05'0000h - 05'FFFFh	64K
B0F9	06'0000h - 06'FFFFh	06'0000h - 06'FFFFh	64K
B1F0	07'0000h - 07'FFFFh	07'0000h - 07'FFFFh	64K
B1F1	08'0000h - 08'FFFFh	08'0000h - 08'FFFFh	64K

Table 4. 320 Kbyte XFlash memory block organization

Block	Addresses (ROMS1='0')	Addresses (ROMS1='1')	Size (bytes)
B2F0	09'0000h - 09'FFFFh	09'0000h - 09'FFFFh	64K
B2F1	0A'0000h - 0A'FFFFh	0A'0000h - 0A'FFFFh	64K
B2F2	0B'0000h - 0B'FFFFh	0B'0000h - 0B'FFFFh	64K
B3F0	0C'0000h - 0C'FFFFh	0C'0000h - 0C'FFFFh	64K
B3F1	0D'0000h - 0D'FFFFh	0D'0000h - 0D'FFFFh	64K
CTRL Registers	0E'0000h - 0E'FFFFh	0E'0000h - 0E'FFFFh	64K

Code fetches are always made on even byte addresses. The last valid code location must contain a branch instruction (unconditional), because sequential boundary crossing from internal Flash to external memory is not supported and causes erroneous results.

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

For PEC data transfers the internal Flash can be accessed independently of the contents of the DPP registers via the PEC source and destination pointers.

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

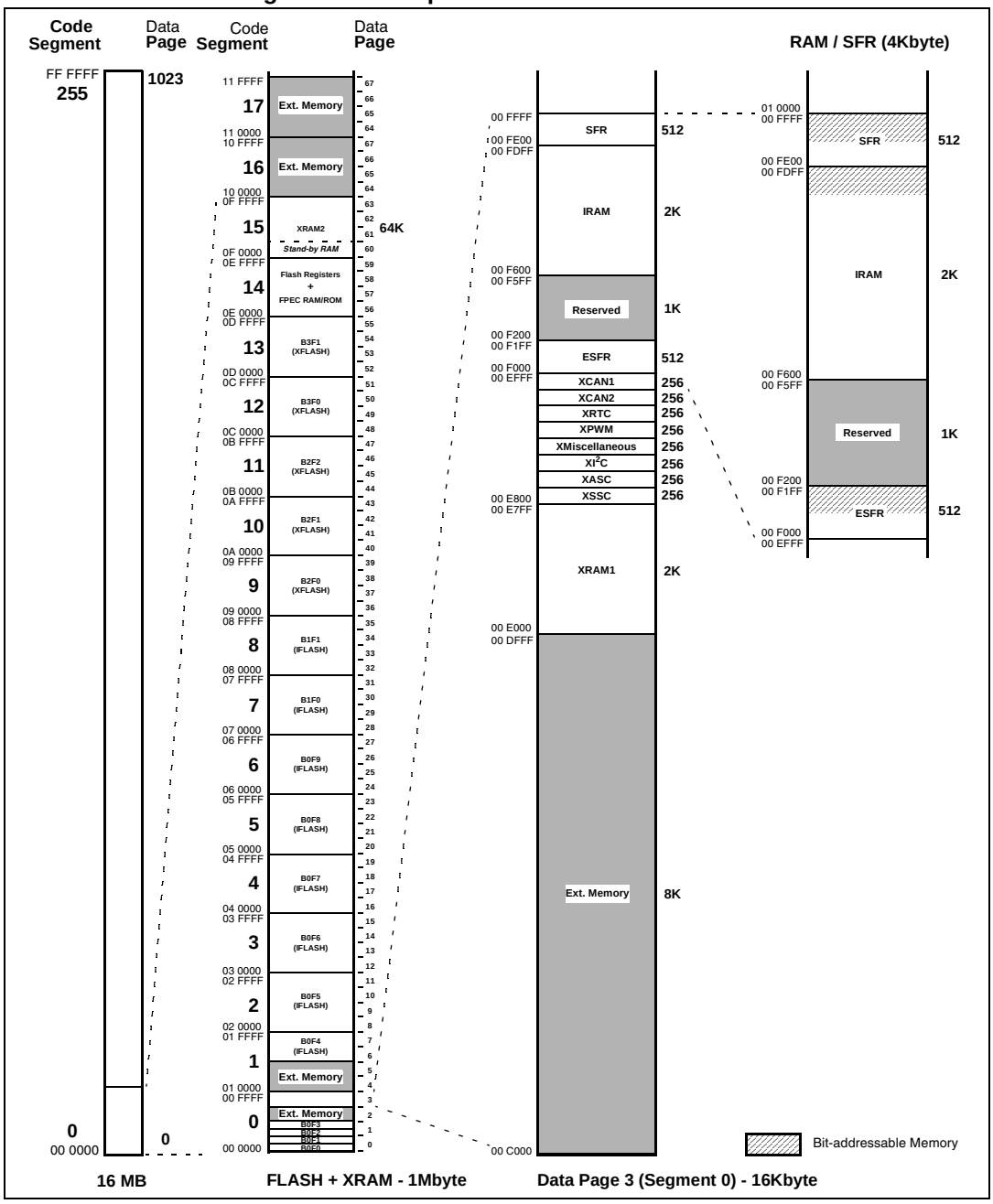
The first 32 Kbytes of the internal Flash may be mapped into segment 0 or segment 1 under software control. *Section 27.10: Handling the internal Flash on page 556* describes the mapping procedures and precautions.

2.3 IRAM and SFR area

The IRAM/SFR area is located within data page 3 and provides access to the 2 Kbyte IRAM (organized as 1K x 16) and to two 512 byte blocks of Special Function Registers (SFRs). The IRAM 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. On-chip RAM and SFR/ESFR areas



Note: 1 The upper 256 bytes of SFR area, ESFR area and IRAM are bit-addressable.

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

Code accesses are always made on even byte addresses. The last valid code location must contain a branch instruction (unconditional), because sequential boundary crossing from IRAM to the SFR area is not supported and can causes erroneous results.

Any word and byte data in the IRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word data access is made on an even byte address. For PEC data transfers, the IRAM can be accessed independently of the contents of the DPP registers via the PEC source and destination pointers.

The upper 256 bytes of the IRAM (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 1 on page 23*).

2.3.1 System stack

The system stack may be defined within the IRAM. The size of the system stack is controlled by bit-field STKSZ in the SYSCON register (see *Table 5*).

For all system stack operations the IRAM is accessed via the Stack Pointer (SP) register. The stack grows downward from higher towards lower IRAM address locations.

Only word accesses are supported by the system stack. A stack overflow (STKOV) and a stack underflow (STKUN) register are provided to control the lower and upper limits of the selected stack area.

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

The technique of implementing this circular stack is described in *Section 27.1: Stack operations on page 547*.

Table 5. Stack size

(STKSZ)	Stack size (words)	IRAM 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)

2.3.2 General purpose registers

The general purpose registers (GPRs) use a block of 16 consecutive words within the IRAM. 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 16 byte GPRs are mapped onto the first eight word GPRs (see *Table 6*).

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

The GPRs are accessed via short 2-, 4- or 8-bit addressing modes using the context pointer (CP) register as base address (independent of the current DPP register contents). Additionally, each bit in the currently active register bank can be accessed individually.

The ST10F276 supports fast register bank (context) switching. Multiple register banks can physically exist within the IRAM 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 IRAM.

Details on using, switching and overlapping register banks are described in Register Banking *Section 27.2: Register banking on page 551*.

Table 6. Mapping of general purpose registers to RAM addresses

IRAM 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

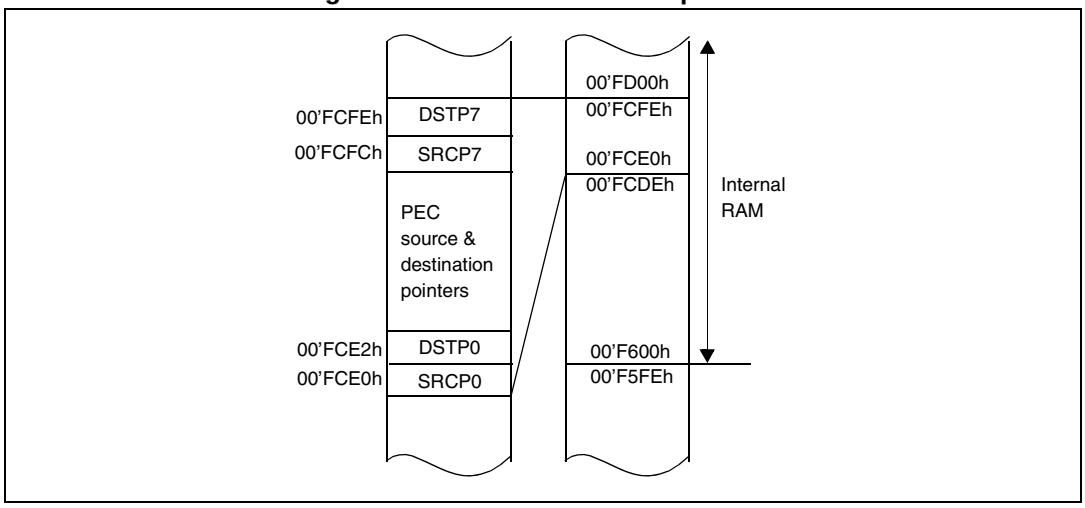
2.3.3 PEC source and destination pointers

The 16 word locations in the IRAM 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 on page 48*).

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 *Section 27: System programming on page 545*.

Figure 7. Location of the PEC pointers



2.3.4 Special function registers

The functions of the CPU, the bus interface, the I/O ports and the on-chip peripherals of the ST10F276 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 IRAM (00'FFFFh...00'FE00h), the second register block, the Extended SFR (ESFR) area, is located in the 512 byte below the IRAM (00'F1FFh...00'F000h).

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

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

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

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

Example:

```
EXTR   #3              ;Switch to ESFR area for the next 3 instructions
MOV    ODP2,           ;this instruction uses 8-bit reg addressing
       #data16         ;(EXTR mandatory)
BSET   DP1h.7          ;this instructiion uses direct bit addressing
                       ;(EXTR mandatory)
```

```
                                ;This instruction uses 16-bit address to access
MOV    T8REL, R1                ;ESFR T8REL. R1 is duplicated and also ;accessible via
                                the ESFR mode
                                ;(EXTR is not required for this access)
;----- ;------                  ;The scope of the EXTR #3 instruction ends here!
MOV    T8REL, R1                ;This instruction uses 16-bit address, and does
                                ;not require switching
```

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

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

2.4 The on-chip XRAM

The 66 Kbytes of on-chip extension RAM (single port XRAM) is provided as a storage for data, user stack and code. It is partially located within data page 3 (XRAM1 - 2 Kbytes), and partially in code segment 15 (XRAM2 - 64 Kbytes).

Both the XRAM modules are connected to the internal XBUS and are accessed like an external memory in 16-bit de-multiplexed bus mode without wait-state or read/write delay (31.25 ns access at 64 MHz CPU clock). Byte and word accesses are allowed.

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 bit VISIBLE in the SYSCON register is set also, accesses to the on-chip XRAM are made visible on the external Port pins. Code fetches are always made on even byte addresses. Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for XRAM operands. Sequential boundary crossing from XRAM to external memory is not supported and causes erroneous results.

The XRAM1 address range is 00’E000h - 00’E7FFh if XPEN (bit 2 of SYSCON register), and XRAM1EN (bit 2 of XPERCON register) are set. If XRAM1EN or XPEN is cleared, then any access in the address range 00’E000h - 00’E7FFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register.

The XRAM2 address range is 0F’0000h - 0F’FFFFh if XPEN (bit 2 of SYSCON register), and XRAM2EN (bit 3 of XPERCON register) are set. If bit XPEN is cleared, then any access in the address range 0F’0000h - 00’FFFFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register; the same occurs when bit XPEN is set, but both bits XRAM2EN and XFLASHEN are cleared.

The lower 16 Kbyte portion of XRAM2 (Address range 0F’0000h-0F’3FFFh) represents the Stand-by RAM, which can be maintained biased through $\overline{EA}/V_{STBY}$ pin when main supply V_{DD} is turned off. As the XRAM appears like external memory, it cannot be used as system stack or as register banks. The XRAM is not provided for single bit storage and therefore is not bit addressable.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for XRAM operands. Any word data access is made to an even byte address. For PEC data transfers XRAM1 can be accessed independently of the contents of the DPP registers, via the PEC source and destination pointers. XRAM2 is not PEC addressable, since not mapped in code segment 0.

The on-chip XRAM is accessed without any wait-states, using 16-bit de-multiplexed bus cycles which takes one instruction cycle. Even if the XRAM is used as external memory, it does not occupy BUSCONx / ADDRSELx registers, but it is selected via additional dedicated XBCON / XADRS registers. In general, these registers are mask-programmed and are not user accessible. The address area of 00'E000h to 00'E7FFh is reserved for XRAM1 accesses, and the address area of 0F'0000h - 0F'FFFFh is reserved for XRAM2 accesses. In ST10F276 the register XADRS3 used for XRAM2 and XFLASH memory range is user programmable: this allows to redefine the size and starting address of the memory window, making possible to play with on-chip and external memory resources (refer to *Section 8.7: The XBUS interface on page 207* for details).

2.4.1 XRAM access via external masters

When bit XPER-SHARE in register SYSCON is set the on-chip XRAM of the ST10F276 can be accessed by an external master during hold mode, via the ST10F276's bus interface. These external accesses must use the same configuration as the internally programmed. No wait-states are required.

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

External accesses to the other XBUS peripherals are not guaranteed in terms of AC Timings. Note that setting XPER-SHARE Mode affects the system configuration: since the bus control functions BREQ, HLDA and HOLD are mapped as alternate functions of P6(7:5), the XSSC module is not accessible when arbitration is in use. For similar reasons, in case segment lines A(23:20) on Port4 have to be used (SALSEL = 10), the CAN1, CAN2 and I^2C modules might not be accessible.

2.5 External memory space

The ST10F276 is capable of using an address space of up to 16 Mbytes. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (Flash) or for registers, may refer to external memory locations. This external memory is accessed via the ST10F276's external bus interface.

Four memory bank sizes are supported:

- Non-segmented mode: 64 Kbytes with A15...A0 on PORT0 or PORT1
- 2-bit segmented mode: 256 Kbytes with A17...A16 on Port4 and A15...A0 on PORT0 or PORT1
- 4-bit segmented mode: 1 Mbyte with A19...A16 on Port4 and A15...A0 on PORT0 or PORT1
- 8-bit segmented mode: 16 Mbytes with A23...A16 on Port4 and A15...A0 on PORT0 or PORT1

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

The ST10F276 also supports **four different bus types**:

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

Memory model and bus mode are selected during reset by pin $\overline{EA}$ and PORT0 pins. For further details about the external bus configuration and control see *Section 8: The external bus interface on page 181*.

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 therefore it is not bit addressable.

2.6 Crossing memory boundaries

The address space of the ST10F276 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 IRAM/SFR area, the internal Flash Memory, the on-chip X-Peripherals (if integrated) and the external memory.

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

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

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

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

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

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

3 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 ST10F276 are executed in one instruction cycle due to this parallelism.

This section 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 *Section 8: The external bus interface on page 181*.

The on-chip peripheral units of the ST10F276 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 erratic behaviour 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:

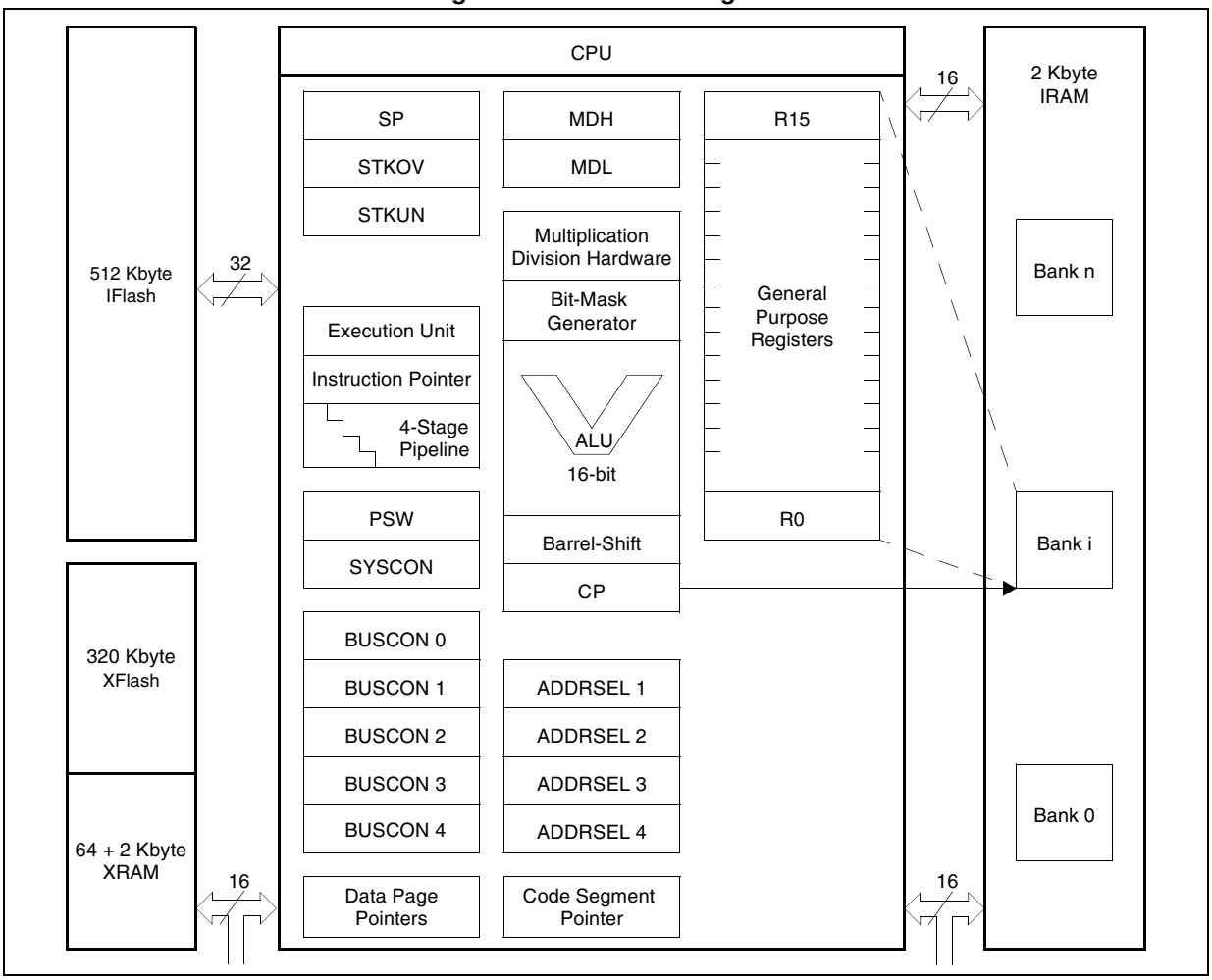
- **RESET state:** Any reset (hardware, software, watchdog) forces the CPU into a pre-defined active state.
- **IDLE state:** The clock signal to the CPU itself is switched off, while the clocks for the on-chip peripherals keep running.
- **POWER DOWN state:** All of the on-chip clocks are switched off.

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 ST10F276 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 8. CPU block diagram



3.1 Instruction pipelines

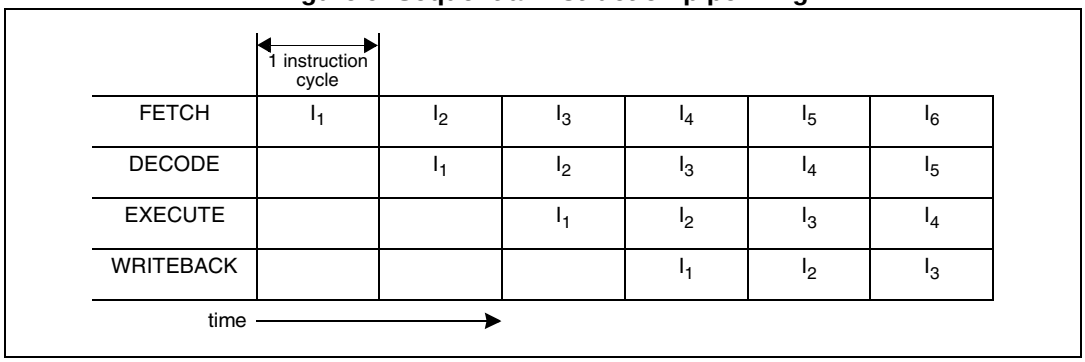
The instruction pipeline breaks down CPU processing into the four following stages:

- Fetch: An instruction selected by the Instruction Pointer (IP) and the Code Segment Pointer (CSP) is fetched from either the internal memory, IRAM, XRAM 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 IRAM 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 9. Sequential instruction pipelining



3.1.1 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 9 on page 55*).

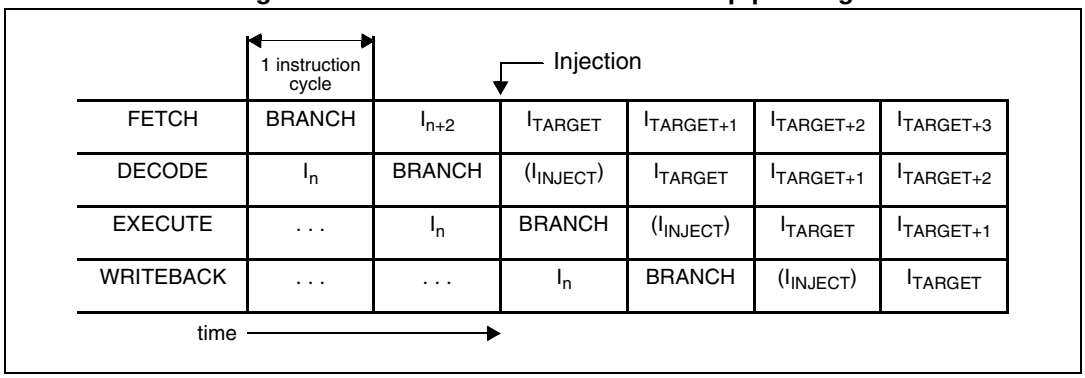
Specification of instruction execution time always refers to the average execution time for pipelined parallel instruction processing (see *Figure 9 on page 55*).

3.1.2 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 10 on page 56*). 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.

Figure 10. Standard branch instruction pipelining



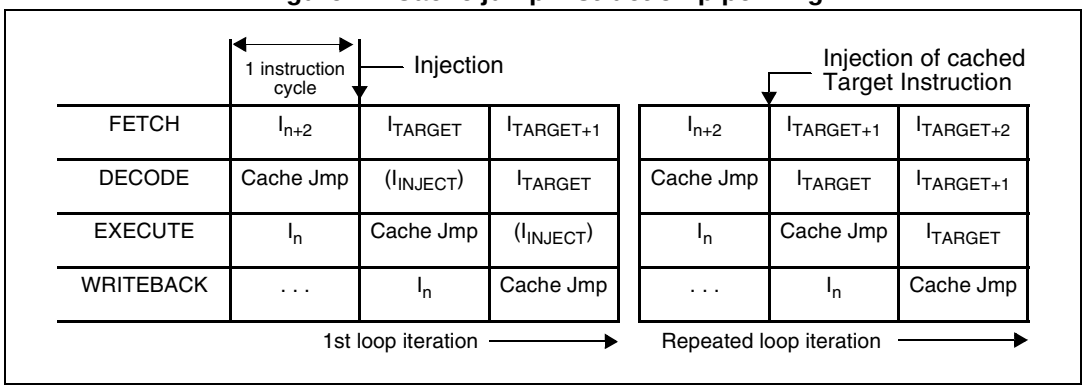
3.1.3 Cache jump instruction processing

The ST10F276 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 sequential instruction is fetched as usual, causing a time delay of one instruction cycle.

If the instruction is repeated in a loop, the target instruction 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 11 on page 56*).

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. Cache jump instruction pipelining



3.1.4 Particular pipeline effects

Since up to four different instructions are processed simultaneously, additional hardware has been included in the ST10F276 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 ST10F276.

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 IRAM 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 into 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.

3.2 Bit-handling and bit-protection

The ST10F276 provides several mechanisms for bit manipulation. These mechanisms, either handle software flags within the IRAM, control on-chip peripherals via control bits in their respective SFRs, or control I/O functions via port pins.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV and BMOVN, explicitly set or clear specific bits. The instructions BFLDL and BFLDH make it possible to change up to 8 bits 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 affect the respective bit location.

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

- Bit can only be modified within the internal specific address areas (IRAM, SFRs...). External locations cannot be used with bit instructions.
- The upper 256 bytes of the SFR area, the ESFR area and the IRAM are bit-addressable (see *Section 2: Memory organization on page 40*). 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 bits. 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 3.1.4: Particular pipeline effects on page 56*).

Protected bits: As mentioned in *Section 1.6: Protected bits on page 39* (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 1.6: Protected bits on page 39).

3.3 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 7* shows the minimum execution times required to process a ST10F276 instruction fetched from the internal IFlash, the IRAM, 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 ST10F276 instructions except some of the branches, the multiplication, the division and a special move instruction. In case of execution from the internal Program Memory, there is no execution time dependency on the instruction length, except for some special branch situations. Because of the short execution time, execution from on-chip RAM (IRAM and XRAM) is flexible for loadable and modifiable code. Execution from external memory depends on the selected bus mode and the

programming of the bus cycles (wait-states). The operand and instruction accesses listed below can extend the execution time of an instruction:

- Internal IFlash Memory operand reads (same for byte and word operand reads),
- Internal IRAM 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 IFlash Memory space,
- Testing Branch Conditions immediately after PSW writes.

Table 7. Minimum execution times

Memory area	Instruction fetch	
	Word instruction (CPU clock cycles)	Double word instruction (CPU clock cycles)
Internal Memory (IFlash)	2	2
Internal Memory (XFlash)	2	4
Internal IRAM	6	8
Internal XRAM	2	4
16-bit De-multiplex Bus	2	4
16-bit Multiplexed Bus	3	6
8-bit De-multiplex Bus	4	8
8-bit Multiplexed Bus	6	12

3.4 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.

Note: 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 bits cannot be modified, and will always supply a read value of '0'.

3.4.1 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 bits).

SYSCON (FF12h / 89h) SFR Reset Value: 0xx0h

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

Reset Value: 0000 0xx0 x000 0000b

Bit	Function
XPER-SHARE	**XBUS Peripheral Share Mode Control** '0': External accesses to XBUS peripherals are disabled. '1': XRAM1 and XRAM2 are accessible via the external bus during hold mode. External accesses to the other XBUS peripherals are not guaranteed in terms of AC timings. See *Section 2.4.1: XRAM access via external masters on page 50* for additional details.
VISIBLE	**Visible Mode Control** '0': Accesses to XBUS peripherals and XRAM are done internally. '1': XBUS peripheral accesses are made visible on the external pins.
XPEN	**XBUS Peripheral Enable bit** '0': Accesses to the on-chip X-peripherals and XRAM are disabled. '1': The on-chip X-peripherals are enabled.
BDRSTEN	**Bidirectional Reset Enable** '0': $\overline{\text{RSTIN}}$ 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 internal reset sequence.
OWDDIS	**Oscillator Watchdog Disable Control** '0': Oscillator Watchdog (OWD) is enabled. If PLL is bypassed, the OWD monitors XTAL1 activity. If there is no activity on XTAL1 for at least 1 µs, the CPU clock is switched automatically to PLL's base frequency (from 750 kHz to 3 MHz). '1': OWD is disabled. If the PLL is bypassed, the CPU clock is always driven by XTAL1 signal. The PLL is turned off to reduce power supply current.

Bit	Function
PWDCFG	**Power Down Mode Configuration Control** '0': Power Down Mode can only be entered during PWRDN instruction execution if NMI pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must 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 (or alternate source see *Section 5.6.1: Fast external interrupts on page 115*) or with external reset.
CSCFG	**Chip Select Configuration Control** '0': Latched Chip Select lines, CSx changes 1 TCL after rising edge of ALE. '1': Unlatched Chip Select lines, CSx changes with rising edge of ALE.
WRCFG	**Write Configuration Control** (Inverted copy of WRC bit of RP0H) '0': Pins WR and BHE retain their normal function. '1': Pin WR acts as WRL, pin BHE acts as WRH.
CLKEN	**System Clock Output Enable** (CLKOUT) '0': CLKOUT disabled, pin may be used for general purpose I/O. '1': CLKOUT enabled, pin outputs the system clock signal or a prescaled value of system clock according to XCLKOUTDIV register setting.
BYTDIS	**Disable/Enable Control for Pin BHE** (Set according to data bus width) '0': Pin BHE enabled. '1': Pin BHE disabled, pin may be used for general purpose I/O.
ROMEN	**Internal Memory Enable** (Set according to pin EA during reset) '0': Internal memory disabled: accesses to the IFlash Memory area use the external bus. '1': Internal memory enabled.
SGTDIS	**Segmentation Disable/Enable Control** '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit). '1': Segmentation disabled (Only IP is saved/restored).
ROMS1	**Internal Memory Mapping** '0': Internal memory area mapped to segment 0 (00'0000h...00'7FFFh). '1': Internal memory area mapped to segment 1 (01'0000h...01'7FFFh).
STKSZ	**System Stack Size** Selects the size of the system stack (in the IRAM) 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 8.2: External bus modes on page 183.

System clock output enable (CLKEN)

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

programmed prescaled value of the f_{CPU} (prescaler factor is programmable from 1 to 256 linearly, default value after reset is 1).

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

Segmentation disable/enable control (SGTDIS)

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

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

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

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

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

System stack size (STKSZ)

This bit-field defines the size of the physical system stack, which is located in the IRAM of the ST10F276. An area of 32...1024 words or all of the IRAM may be dedicated to the system stack. A so-called "circular stack" mechanism allows to use a bigger virtual stack than this dedicated IRAM area. These techniques as well as the encoding of bit-field STKSZ are described in more detail in Stack Operations (see *Section 27.1: Stack operations on page 547*).

Table 8. Stack size

(STKSZ)	Stack size (words)	IRAM 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)

3.4.2 X-Peripherals control register (XPERCON)

XPERCON (F024h / 12h) ESFR Reset Value: - 005h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	XMISC EN	XI2C EN	XSSC EN	XASC EN	XPWM EN	XFLAS HEN	XRTC EN	XRAM2 EN	XRAM1 EN	CAN2 EN	CAN1 EN
					RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
CAN1EN	**CAN1 Enable Bit** ‘0’: Accesses to the on-chip CAN1 XPeripheral and its functions are disabled (P4.5 and P4.6 pins can be used as general purpose IOs, but address range 00’EF00h-00’EFFFh is directed to external memory only if CAN2EN, XRTCEN, XASCEN, XSSCEN, XI2CEN, XPWMEN an XMISCEN are ‘0’ also). ‘1’: The on-chip CAN1 XPeripheral is enabled and can be accessed.
CAN2EN	**CAN2 Enable Bit** ‘0’: Accesses to the on-chip CAN2 XPeripheral and its functions are disabled (P4.4 and P4.7 pins can be used as general purpose IOs, but address range 00’EE00h-00’EEFFh is directed to external memory only if CAN1EN, XRTCEN, XASCEN, XSSCEN, XI2CEN, XPWMEN and XMISCEN are ‘0’ also). ‘1’: The on-chip CAN2 XPeripheral is enabled and can be accessed.
XRAM1EN	**XRAM1 Enable Bit** ‘0’: Accesses to the on-chip 2 Kbyte XRAM are disabled. Address range 00’EE00h-00’EEFFh is directed to external memory. ‘1’: The on-chip 2 Kbyte XRAM is enabled and can be accessed.
XRAM2EN	**XRAM2 Enable Bit** ‘0’: Accesses to the on-chip 64 Kbyte XRAM are disabled, external access performed. Address range 0F’0000h-0F’FFFFh is directed to external memory only if XFLASHEN is ‘0’ also. ‘1’: The on-chip 64 Kbyte XRAM is enabled and can be accessed.
XRTCEN	**RTC Enable** ‘0’: Accesses to the on-chip RTC module are disabled, external access performed. Address range 00’ED00h-00’EDFF is directed to external memory only if CAN1EN, CAN2EN, XASCEN, XSSCEN, XI2CEN, XPWMEN and XMISCEN are ‘0’ also. ‘1’: The on-chip RTC module is enabled and can be accessed.
XPWMEN	**XPWM Enable** ‘0’: Accesses to the on-chip XPWM module are disabled, external access performed. Address range 00’EC00h-00’ECFF is directed to external memory only if CAN1EN, CAN2EN, XASCEN, XSSCEN, XI2CEN, XRTCEN and XMISCEN are ‘0’ also. ‘1’: The on-chip XPWM module is enabled and can be accessed.
XFLASHEN	**XFlash Enable Bit** ‘0’: Accesses to the on-chip XFlash and Flash control registers are disabled, external access performed. Address range 09’0000h-0E’FFFFh is directed to external memory only if XRAM2EN is ‘0’ also. ‘1’: The on-chip XFlash is enabled and can be accessed.

Bit	Function
XASCEN	**XASC Enable Bit** '0': Accesses to the on-chip XASC are disabled, external access performed. Address range 00'E900h-00'E9FFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XI2CEN, XPWMEN and XMISCEN are '0' also. '1': The on-chip XASC is enabled and can be accessed.
XSSCEN	**XSSC Enable Bit** '0': Accesses to the on-chip XSSC are disabled, external access performed. Address range 00'E800h-00'E8FFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XI2CEN, XPWMEN and XMISCEN are '0' also. '1': The on-chip XSSC is enabled and can be accessed.
XI2CEN	**I^2C Enable Bit** '0': Accesses to the on-chip I^2C are disabled, external access performed. Address range 00'EA00h-00'EAFFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XSSCEN, XPWMEN and XMISCEN are '0' also. '1': The on-chip I2C is enabled and can be accessed.
XMISCEN	**XBUS Additional Features Enable Bit** '0': Accesses to the Additional Miscellaneous Features is disabled. Address range 00'EB00h-00'EBFFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XSSCEN, XPWMEN and XI2CEN are '0' also. '1': The Additional Features are enabled and can be accessed.

When CAN1, CAN2, RTC, XASC, XSSC, I^2C, XPWM and the XBUS Additional Features are all disabled via XPERCON setting, then any access in the address range 00'E800h - 00'EFFFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register. All pins involved with X-Peripherals, can be used as General Purpose IO whenever the related module is not enabled.

The default X-Peripheral configuration after reset is such that only CAN1 and XRAM1 are pre-selected: they will be enabled once XPEN bit in SYSCON register is set.

Register XPERCON cannot be changed after the global enabling of X-Peripherals, that is, after setting of bit XPEN in SYSCON register.

In Emulation mode, all the X-Peripherals are enabled (XPERCON bits are all set). It is up to the emulation device to redirect or not an access to external memory or to XBUS. Register XPEREMU has been created to allow a dynamic selection of this redirection instead of a static configuration of the emulator at the start-up.

Reserved bits of XPERCON register should always be written to '0'.

3.4.3 XPERCON and XPEREMU registers

As already mentioned, XPERCON register has to be programmed to enable the single XBUS modules separately. The XPERCON is a read/write ESFR register; the XPEREMU register is a write-only register mapped on XBUS memory space (address EB7Eh).

Once the XPEN bit of SYSCON register is set and at least one of the X-peripherals (except memories) is activated, the register XPEREMU must be written with the same content of XPERCON: this is mandatory in order to allow a correct emulation of the new set of features

introduced on XBUS for the new ST10 generation. The following instructions must be added inside the initialization routine:

```
if (SYSCON.XPEN && (XPERCON & 0x07D3))
then { XPEREMU = XPERCON }
```

Of course, XPEREMU must be programmed after XPERCON and after SYSCON, in such a way the final configuration for X-Peripherals is stored in XPEREMU and used for the emulation hardware setup.

XPEREMU (EB7Eh) XBUS Reset Value: xxxxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	XMISC EN	XI2C EN	XSSC EN	XASC EN	XPWM EN	XFLAS HEN	XRTC EN	XRAM2 EN	XRAM1 EN	CAN2 EN	CAN1 EN
					W	W	W	W	W	W	W	W	W	W	W

The bit meaning is exactly the same as XPERCON.

3.4.4 Emulation dedicated registers

Four additional registers are implemented for emulation purpose only. Similarly to XPEREMU, they are write only registers and reserved for emulator software usage. User should not write to these registers.

XEMU0 (EB76h) XBUS Reset Value: xxxxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XEMU0(15:0)															
W															

XEMU1 (EB78h) XBUS Reset Value: xxxxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XEMU1(15:0)															
W															

XEMU2 (EB7Ah) XBUS Reset Value: xxxxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XEMU2(15:0)															
W															

XEMU3 (EB7Ch) XBUS Reset Value: xxxxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XEMU3(15:0)															
W															

3.4.5 The processor status word PSW

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

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

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

Bit	Function
N	**Negative Result** Set when the result of an ALU operation is negative.
C	**Carry Flag** Set when the result of an ALU operation produces a carry bit.
V	**Overflow Result** Set when the result of an ALU operation produces an overflow.
Z	**Zero Flag** Set when the result of an ALU operation is zero.
E	**End of Table Flag** Set when the source operand of an instruction is 8000h or 80h.
MULIP	**Multiplication/Division In Progress** ‘0’: There is no multiplication/division in progress. ‘1’: A multiplication/division has been interrupted.
USR0	**User General Purpose Flag** May be used by the application software.
HLDEN, ILVL, IEN	**Interrupt and EBC Control Fields** Define the response to interrupt requests and enable external bus Arbitration (Described in *Section 5: Interrupt and trap functions on page 96*).

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 bits are cleared.

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

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

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

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

V-Flag: For addition, subtraction and 2's complementation the V-flag is always set to '1' if the result overflows the maximum range of signed numbers, which are representable by either 16-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 *Table 9*). For Boolean bit operations with only one operand the V-flag is always cleared. For Boolean bit operations with two operands the V-flag represents the logical ORing of the two specified bits.

Table 9. Shift right rounding error evaluation

C-flag	V-flag	Rounding error quantity
0	0	No rounding error
0	1	0 <Rounding error< $^1/_2$ LSB
1	0	Rounding error= $^1/_2$ LSB
1	1	Rounding error> $^1/_2$ LSB

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

For Boolean bit operations with only one operand the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands the Z-

flag represents the logical NORing of the two specified bits. For the prioritize ALU operation the Z-flag indicates, if the second operand was zero or not.

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

If the value of the source operand of an instruction equals the lowest negative number, which is representable by the data format of the corresponding instruction ('8000h' for the word data type, or '80h' for the byte data type), the E-flag is set to '1', otherwise it is cleared.

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

Note: The MULIP flag is a part of the task environment. When the interrupting service routine does not return to the interrupted multiply/divide instruction (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 4-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 refer to *Section 5: Interrupt and trap functions on page 96*.

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

3.4.6 The instruction pointer IP

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

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

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

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

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

Bit	Function
IP	**Instruction Pointer** Specifies the intra segment offset, from where the current instruction is to be fetched. IP refers to the current segment (SEGNR bit field of CSP register).

3.4.7 The code segment pointer CSP

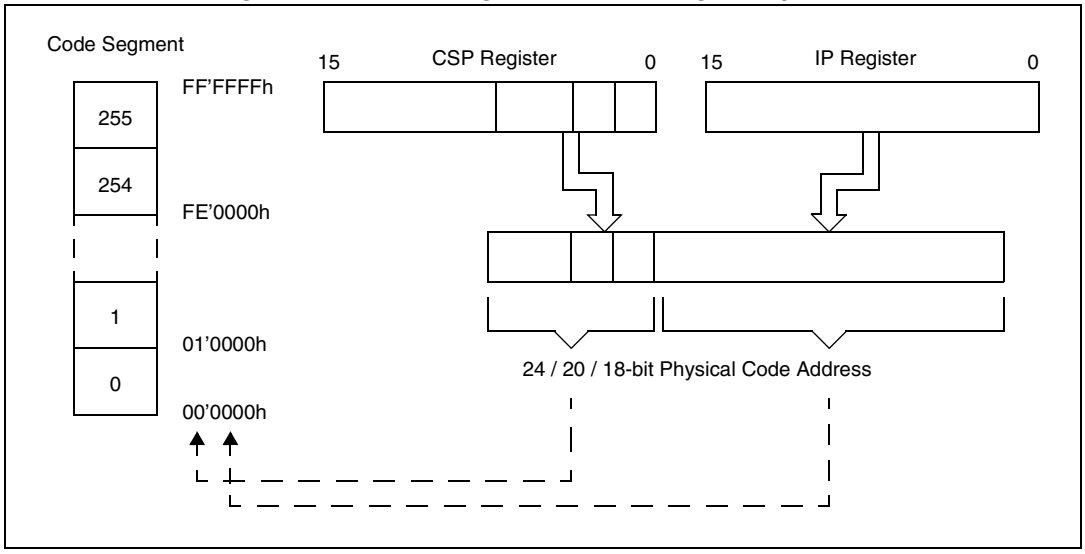
This non-bit-addressable register selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up to 256 segments of 64 Kbytes each, while the upper 8 bits are reserved for future use.

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

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

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

Figure 12. 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 1 on page 23*.

In case of the segmented memory mode the selected number of segment address bits (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.

3.4.8 The data page pointers DPP0, DPP1, DPP2, DPP3

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

The DPP registers are implicitly used whenever data accesses to any memory location are made via indirect, or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers). After reset, the Data Page Pointers are initialized in a way that all indirect or direct long 16-bit addresses result in identical 18-bit addresses. This allows to access data pages 3...0 within segment 0 as shown in the *Figure 13 on page 73*. If the user does not want to use any data paging, no further action is required.

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

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

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

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

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

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

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

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

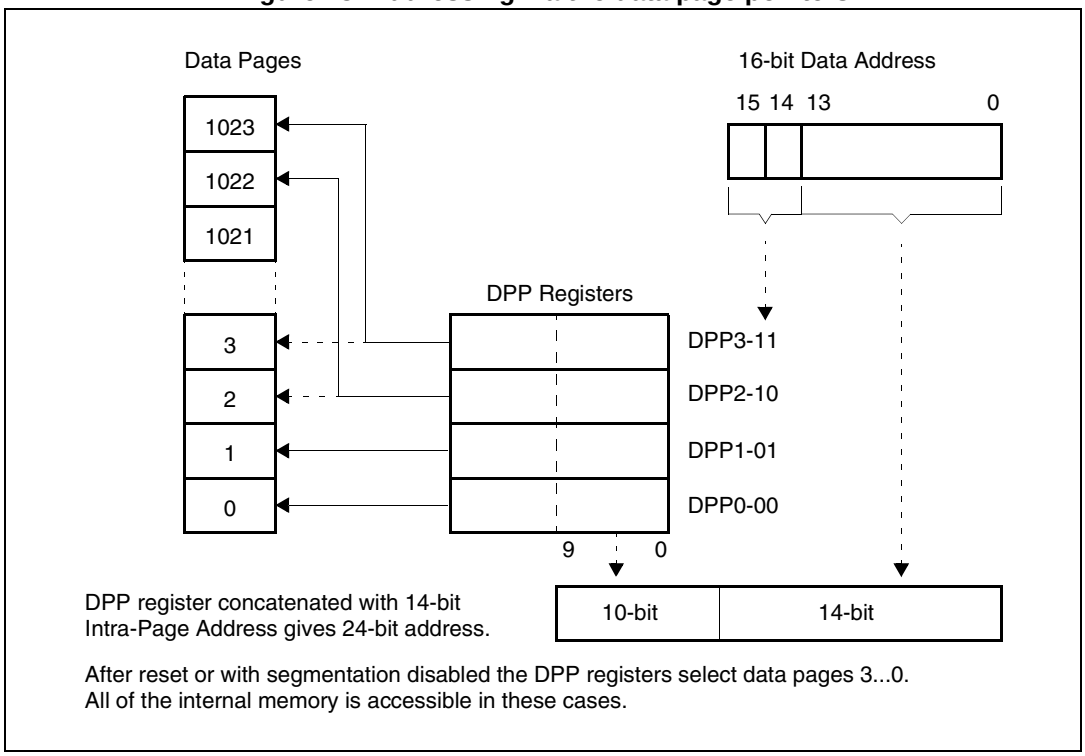
Bit	Function
DPPxPN	**Data Page Number of DPPx** Specifies the data page selected via DPPx. Only the 2 least significant bits of DPPx are used when segmentation is disabled.

Data paging is performed by concatenating the lower 14 bits of an indirect or direct, long 16 bit address with the contents of the DPP register, selected by the upper two bits of the 16-bit address. The content of the selected DPP register specifies one of 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 bits of the implicitly selected DPP register are used to generate the physical address. Thus, extreme care should be taken when changing the content of a DPP register, if a non-segmented memory model is selected, because unexpected results could occur.

In case of the segmented memory mode the selected number of segment address bits (9...2, 5...2 or 3...2) of the respective DPP register is output on the segment address pins A23/A19/A17/A16 of 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 13. Addressing via the data page pointers



3.4.9 The context pointer CP

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

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

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

Bit	Function
CP	**Modifiable portion of register CP** Specifies the (word) base address of the current register bank. When writing a value to register CP with bit CP.11...CP.9 = '000', bit CP.11...CP.10 are set to '11' by hardware, in all other cases all bits 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 IRAM location. If this condition is not met, unexpected results may occur.

- Do not set CP below the IRAM start address, 00'F600h (2 Kbytes).
- Do not set CP above 00'FDFEh.
- Be careful using the upper GPRs with CP above 00'FDE0h.

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

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

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

Several addressing modes use register CP implicitly for address calculations.

Short 4-bit GPR addresses (mnemonic: Rw or Rb) specify an address relative to the memory location specified by the contents of the CP register, which is the base of the current register bank.

Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short 4-bit GPR address is either multiplied by two or not before it is added to the content of register CP (see *Figure 15 on page 75*).

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 bits as short 4-bit GPR address, while the four most significant bits are ignored.

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

Figure 14. Register bank selection via register CP

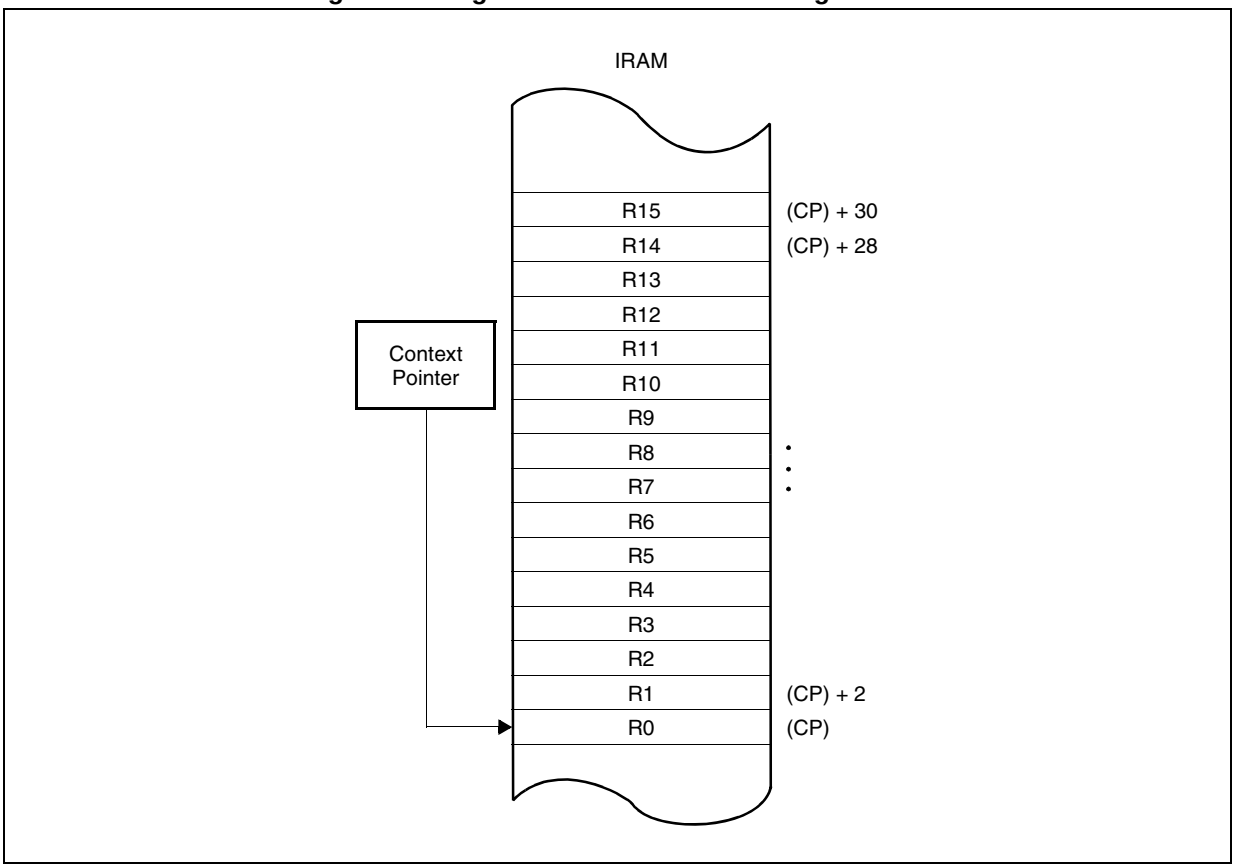
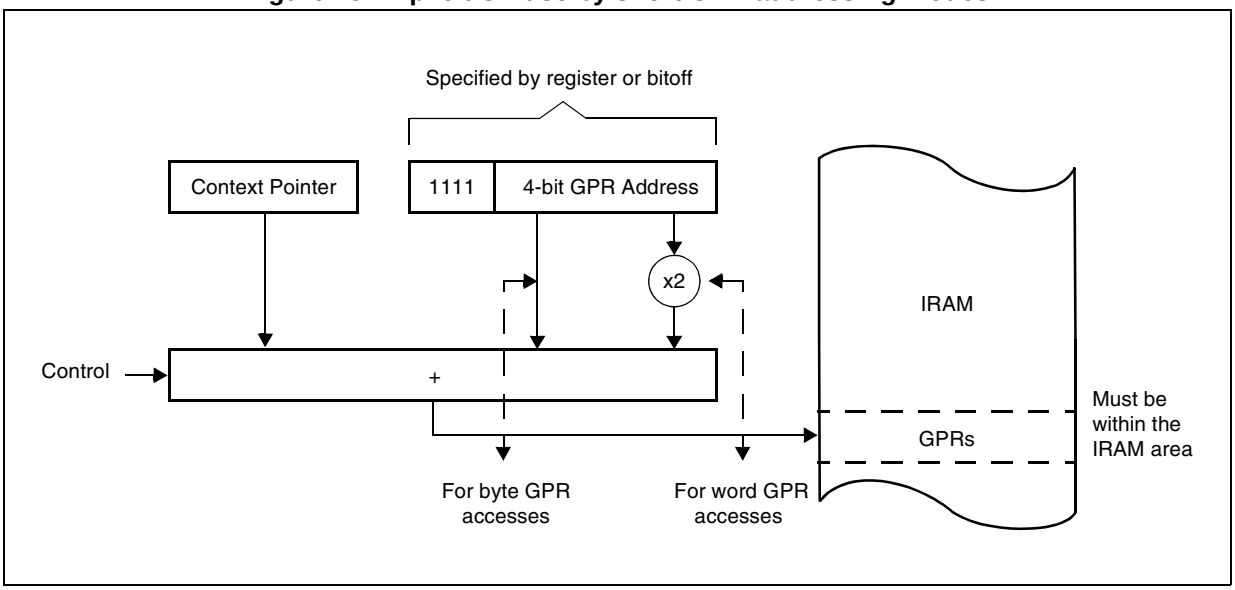


Figure 15. Implicit CP use by short GPR addressing modes



3.4.10 The stack pointer SP

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

Since the least significant bit of register SP is tied to '0' and bits 15 through 12 are tied to '1' by hardware, the SP register can only contain values from F000h to FFFEh. This allows to access a physical stack within the IRAM 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 instructions must not immediately follow an instruction updating the SP register.

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

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

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

3.4.11 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 bits 15 through 12 are tied to '1' by hardware, the STKOV register can only contain values from F000h to FFFEh.

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

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

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

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

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

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

More details about the stack overflow trap service routine and virtual stack management are given in *Section 27: System programming on page 545*.

3.4.12 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 bits 15 through 12 are tied to '1' by hardware, the STKUN register can only contain values from F000h to FFFEh.

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

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

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

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

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

More details about the stack underflow trap service routine and virtual stack management are given in *Section 27: System programming on page 545*.

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.

3.4.13 The multiply / divide high register MDH

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

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

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

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

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

A detailed description of how to use the MDH register for programming multiply and divide algorithms can be found in *Section 27: System programming on page 545*.

3.4.14 The multiply / divide low register MDL

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

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

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

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

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

A detailed description of how to use the MDL register for programming multiply and divide algorithms can be found in *Section 27: System programming on page 545*.

3.4.15 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
MS	**Internal Machine Status** The multiply/divide unit uses these bits to control internal operations. Never modify these bits without saving and restoring register MDC.
MDRIU	**Multiply/Divide Register In Use** ‘0’: Cleared, when register MDL is read via software. ‘1’:Set when register MDL or MDH is written via software, or when a multiply or divide instruction is executed.

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, preparing 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 *Section 27: System programming on page 545*.

3.4.16 The constant zeros register ZEROS

All bits of this bit-addressable register are fixed to '0' by hardware. This register is read only. Register ZEROS can be used as a register-addressable constant of all zeros, 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

3.4.17 The constant ones register ONES

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

Example

Mask for FFFFh values used 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
```

4 Multiply-accumulate unit (MAC)

The MAC is a specialized co-processor added to the ST10F276 CPU core to improve the performance of signal processing algorithms. It includes:

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

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

4.1 MAC features

Enhanced addressing capabilities

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

General

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

The working register of the MAC Unit is a dedicated 40-bit wide Accumulator register. A set of consistent flags is automatically updated in the MSW register (see *Section 4.3.2: Accumulator & control registers on page 91*) after each MAC operation. These flags allow branching on specific conditions. Unlike the PSW flags, these flags are not preserved automatically by the CPU upon entry into an interrupt or trap routine.

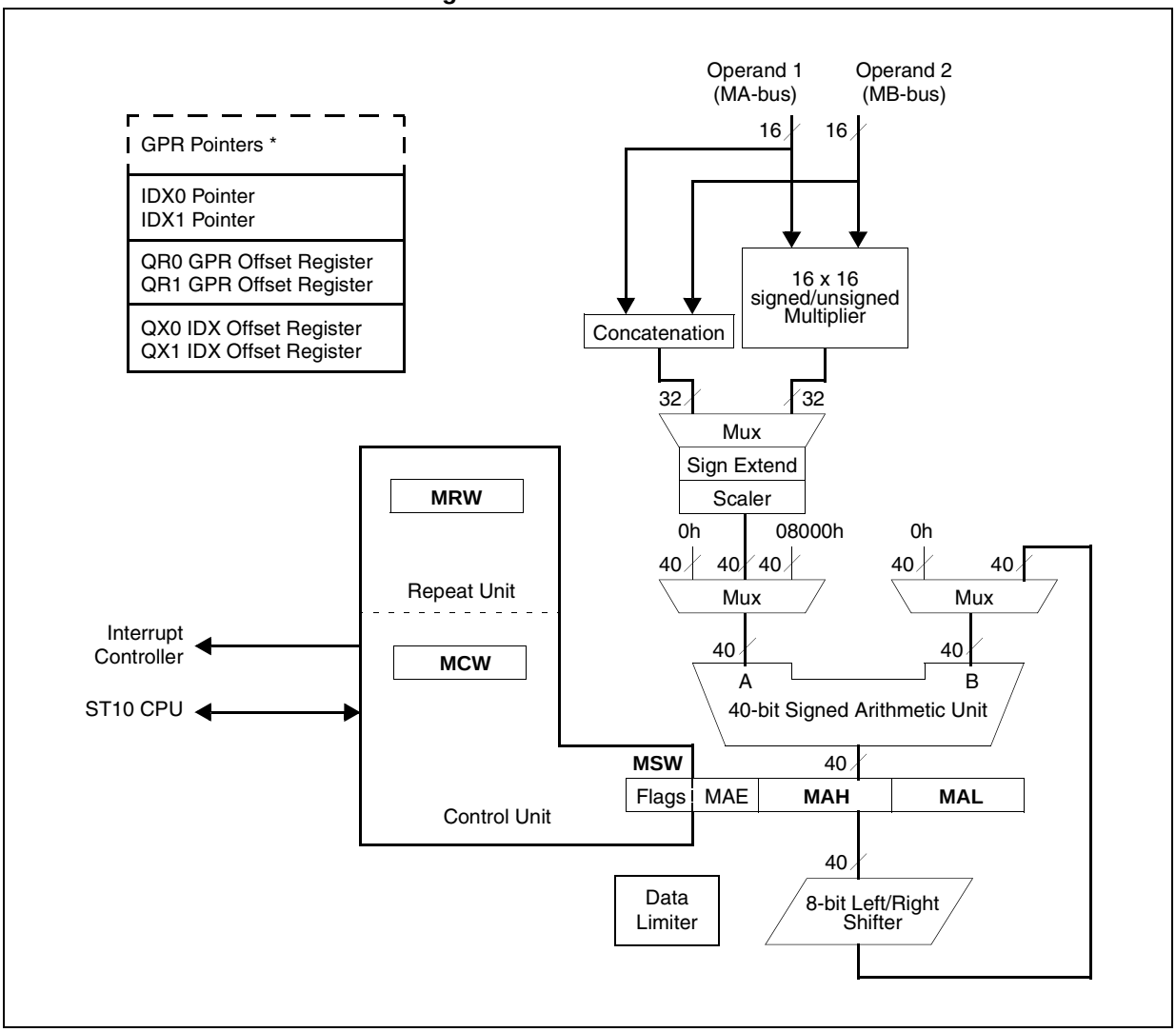
Note: ***All dedicated MAC registers must be saved on the stack if the MAC unit is shared between different tasks and interrupts.***

Program control

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

4.2 MAC operation

Figure 16. MAC architecture



1. * Shared with standard ALU.

4.2.1 Instruction pipelining

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

- FETCH: All new instructions are double-word instructions.
- DECODE: If required, operand addresses are calculated and the resulting operands are fetched. IDX and GPR pointers are post-modified if necessary.
- EXECUTE: Performs the MAC operation. At the end of the cycle, the Accumulator and the MAC condition flags are updated if required. Modified GPR pointers are written-back during this stage, if required.
- WRITEBACK: Operand write-back in the case of parallel data move.

4.2.2 Particular pipeline effects with the MAC unit

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

Initialization of the pointers and offset registers

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

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

Same requirements between the update of one of the offset reg. QX_i & QR_i and their next use.

Read Access to MAC registers (CoReg)

At least one instruction which does not use the MAC must be inserted between a MAC instruction (CoXXX) writing to a MAC register (MAH, MAL, MSW, MRW, MCW) and a standard instruction reading this register. This is because the accumulator and the status of the MAC are modified during the execute stage.

Example 1

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

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

4.2.3 Address generation

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

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

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

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

Table 10. Pointer post-modification combinations for IDXi and Rwn

Symbol	Mnemonic	Address pointer operation
"[$IDX_i\otimes$]" stands for	[IDX_i]	(IDX_i) ← (IDX_i) (no-op)
	[IDX_i+]	(IDX_i) ← (IDX_i) +2 (i = 0,1)
	[IDX_i -]	(IDX_i) ← (IDX_i) -2 (i = 0,1)
	[IDX_i + QX_j]	(IDX_i) ← (IDX_i) + (QX_j) (i, j = 0,1)
	[IDX_i - QX_j]	(IDX_i) ← (IDX_i) - (QX_j) (i, j = 0,1)
"[$Rw_n\otimes$]" stands for	[Rw_n]	(Rw_n) ← (Rw_n) (no-op)
	[Rw_n+]	(Rw_n) ← (Rw_n) +2 (n = 0-15)
	[Rw_n-]	(Rw_n) ← (Rw_n) -2 (k = 0-15)
	[Rw_n+QR_j]	(Rw_n) ← (Rw_n) + (QR_j) (n = 0-15; j = 0,1)
	[Rw_n - QR_j]	(Rw_n) ← (Rw_n) - (QR_j) (n = 0-15; j = 0,1)

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

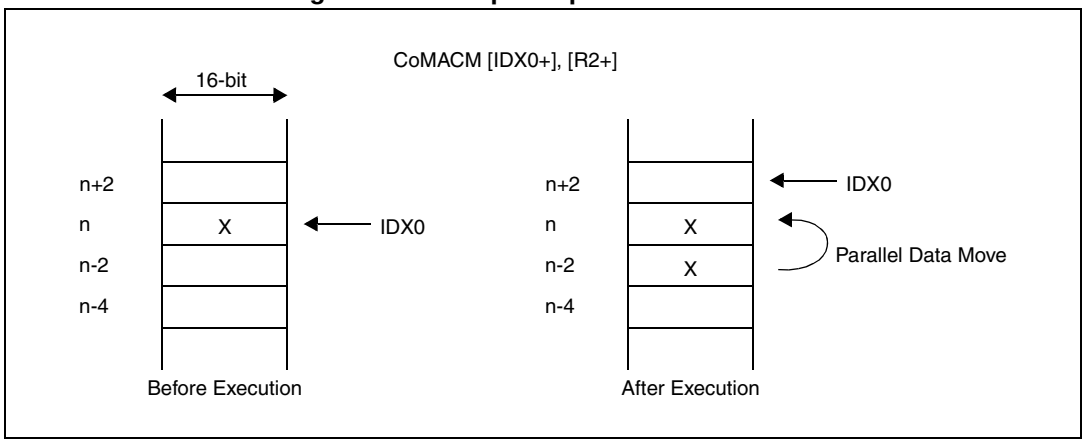
the post-modification of IDX_i. It is obtained by the reverse operation than the one used to calculate the new value of IDX_i. The following table shows these rules.

Table 11. Parallel data move addressing

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

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

Figure 17. Example of parallel data move



4.2.4 16 x 16 signed/unsigned parallel multiplier

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

4.2.5 40-bit signed arithmetic unit

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

The 40-bit arithmetic unit has two 40-bit input ports A and B. The A-input port accepts data from four possible sources: 00'0000'0000h, 00'0000'8000h (round), the sign-extended product, or the sign-extended data conveyed by the 32-bit bus resulting from the

concatenation of MA- and MB-buses. Product and Concatenation can be shifted left by one according to MP for the multiplier or to the instruction for the concatenation. The B-input port is fed either by the 40-bit shifted/not shifted and inverted/not inverted accumulator or by 00'0000'0000h. A-input and B-input ports can receive 00'0000'0000h to allow direct transfers from the B-source and A-source, respectively, to the Accumulator (case of Multiplication and Shift). The output of the arithmetic unit goes to the Accumulator.

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

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

4.2.6 The 40-bit signed accumulator register

The 40-bit accumulator consists of three smaller registers, MAL, MAH, and MAE. MAH and MAL are 16-bit wide, MAE is 8-bit wide. MAE is the most significant byte (MSB) of the 40-bit accumulator, however it is accessed as the least significant byte (LSB) of the MSW register and performs guarding function.

On MAH write operations, the value of the accumulator is automatically adjusted to signed extended 40-bit format. This means:

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

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

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

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

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

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

```
POP MAL
POP MSW
```

4.2.7 The 40-bit adder / subtracter

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

The output of the Adder/Subtracter goes to the 40-bit Signed Accumulator. It is also possible to round and to saturate the result to 32-bit automatically after every accumulation before to be loaded into the accumulator. The round operation is performed by adding 00'0000'8000h to the result. Automatic saturation is enabled by setting the MCW.MS saturation bit.

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

4.2.8 Data limiter

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

Table 12. Limiter output using CoSTORE instruction

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

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

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

4.2.9 The accumulator shifter

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

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

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

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

4.2.10 Repeat unit

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

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

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

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

```
2       MOV MRW, #00FFh           ; load MRW with 255
        NOP                       ; instruction latency
        Repeat MRW times
        CoMACM [IDX1-],[R2+]      ; repeated 256 times
```

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

Repeat sequences may be interrupted. When an interrupt occurs during a repeat sequence, the sequence is stopped and the interrupt routine is executed. The repeat sequence resumes at the end of the interrupt routine. During the interrupt, MR remains set, indicating that a repeated instruction has been interrupted and the Repeat Count holds the number (minus 1) of repetition that remains to complete the sequence. If the Repeat Unit is used in

the interrupt routine, MRW must be saved by the user and restored before the end of the interrupt routine.

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

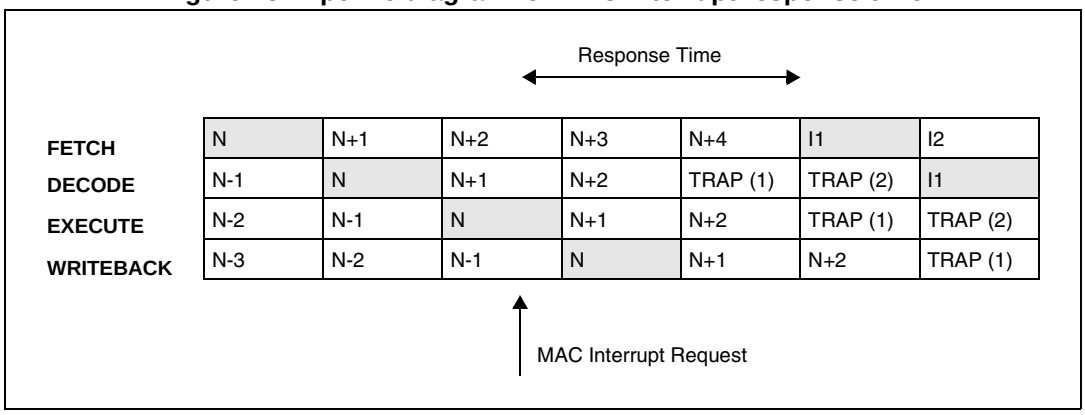
4.2.11 MAC interrupt

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

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

As the MAC status flags are updated (or eventually written by software) during the Execute stage of the pipeline, the response time of a MAC interrupt request is three instruction cycles (see *Figure 18*). It is the number of instruction cycles required between the time the request is sent and the time the first instruction located at the interrupt vector location enters the pipeline. Note that the IP value stacked after a MAC interrupt does not point to the instruction that triggers the interrupt.

Figure 18. Pipeline diagram for MAC interrupt response time



4.2.12 Number representation & rounding

The MAC supports the 2's-complement representation of binary numbers. In this format, the sign bit is the MSB of the binary word. This is set to zero for positive numbers and set to one for negative numbers. Unsigned numbers are supported only by multiply/multiply-accumulate instructions which specifies whether each operand is signed or unsigned.

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

The MAC implements ‘2’s complement rounding’. With this rounding type, one is added to the bit to the right of the rounding point (bit 15 of MAL), before truncation (MAL is cleared).

4.3 MAC register set

4.3.1 Address registers

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

IDX0 (FF08h / 84h) SFR Reset Value: 0000h
IDX1 (FF0Ah / 85h) SFR Reset Value: 0000h

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

Bit	Function
IDXy	16-bit IDXy address (y = 0, 1)

QX0 (F000h / 00h) ESFR Reset Value: 0000h
QX1 (F002h / 01h) ESFR Reset Value: 0000h
QR0 (F004h / 02h) ESFR Reset Value: 0000h
QR1 (F006h / 03h) ESFR Reset Value: 0000h

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

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

4.3.2 Accumulator & control registers

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

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

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

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

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

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

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

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

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

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

Bit	Function
MAE	**Accumulator Extension** (bits [39:32])
N	**Negative Flag** Set when the Accumulator is negative at the end of a MAC operation.
Z	**Zero Flag** Set when the Accumulator is zero at the end of a MAC operation.
C	**Carry Flag** Set when a MAC operation produces a carry or a borrow bit.
SV	**Sticky Overflow Flag** Set when a MAC operation produces a 40-bit arithmetic overflow. It remains set until it is explicitly reset by software.
E	**Extension Flag** Set when MAE contains significant bits at the end of a MAC operation
SL	**Sticky Limit Flag** Set when the result of a MAC operation is automatically saturated. Also used for CoMIN, CoMAX instructions to indicate that the Accumulator has changed. It remains set until it is explicitly reset by software.
MIR	**MAC Interrupt Request** Set when the MAC Unit generates an interrupt request.

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

MCW (FFDCh / EEh) SFR Reset Value: 0000h

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

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

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

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

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

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

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

Table 13. MAC register address in CoReg addressing mode

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

4.4 MAC instruction set summary

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

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

Table 14. MAC instruction set summary

Mnemonic	Addressing modes	Repeatable
CoMUL(u,s,-,rnd)	Rw_n, Rw_m [IDX_i⊗], [Rw_m⊗] Rw_n, [Rw_m⊗]	No
CoMAC(u, s, -, rnd) CoMACR(u, s, rnd)	Rw_n, Rw_m [IDX_i⊗], [Rw_m⊗] Rw_n, [Rw_m⊗]	No Yes Yes
CoMACM (u, s, -, rnd) CoMACMR(u, s, rnd)	[IDX_i⊗], [Rw_m⊗]	Yes
CoNOP	[Rw_m⊗] [IDX_i⊗] [IDX_i⊗], [Rw_m⊗]	Yes
CoNEG CoNEG, rnd CoRND	-	No
CoSTORE	Rw_n, CoReg [Rw_n⊗], Coreg	No Yes
CoMOV	[IDX_i⊗], [Rw_m⊗]	Yes
CoADD(2) CoSUB(2) CoSUB(2)R CoMax CoMin	Rw_n, Rw_m [IDX_i⊗], [Rw_m⊗] Rw_n, [Rw_m⊗]	No Yes Yes

Table 14. MAC instruction set summary (continued)

Mnemonic	Addressing modes	Repeatable
CoLOAD(2, -) CoCMP	Rw_n, Rw_m [IDX_i⊗], [Rw_m⊗] Rw_n, [Rw_m⊗]	No
CoSHL CoSHR CoASHR CoASHR, rnd	Rw_m #data4 [Rw_m⊗]	Yes No Yes
CoABS	- Rw_n, Rw_m [IDX_i⊗], [Rw_m⊗] Rw_n, [Rw_m⊗]	No

5 Interrupt and trap functions

The architecture of the ST10F276 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 ST10F276'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 two 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 ST10F276 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 ST10F276 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 ST10F276, 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 ST10F276 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 ST10F276'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 15* lists all sources that are capable of requesting interrupt or PEC service in the ST10F276, 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 double-word instruction. The respective vector location results from multiplying the trap number by 4 (4 bytes per entry).

Table 15. 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

Table 15. 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 13	CC13IR	CC13IE	CC13INT	00'0074h	1Dh
CAPCOM Register 14	CC14IR	CC14IE	CC14INT	00'0078h	1Eh
CAPCOM Register 15	CC15IR	CC15IE	CC15INT	00'007Ch	1Fh
CAPCOM Register 16	CC16IR	CC16IE	CC16INT	00'00C0h	30h
CAPCOM Register 17	CC17IR	CC17IE	CC17INT	00'00C4h	31h
CAPCOM Register 18	CC18IR	CC18IE	CC18INT	00'00C8h	32h
CAPCOM Register 19	CC19IR	CC19IE	CC19INT	00'00CCh	33h
CAPCOM Register 20	CC20IR	CC20IE	CC20INT	00'00D0h	34h
CAPCOM Register 21	CC21IR	CC21IE	CC21INT	00'00D4h	35h
CAPCOM Register 22	CC22IR	CC22IE	CC22INT	00'00D8h	36h
CAPCOM Register 23	CC23IR	CC23IE	CC23INT	00'00DCh	37h
CAPCOM Register 24	CC24IR	CC24IE	CC24INT	00'00E0h	38h
CAPCOM Register 25	CC25IR	CC25IE	CC25INT	00'00E4h	39h
CAPCOM Register 26	CC26IR	CC26IE	CC26INT	00'00E8h	3Ah
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

Table 15. Interrupt and PEC service request sources (continued)

Source of interrupt or PEC service request	Request flag	Enable flag	Interrupt vector	Vector location	Trap number
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
See *Section 5.7*	XP0IR	XP0IE	XP0INT	00'0100h	40h
See *Section 5.7*	XP1IR	XP1IE	XP1INT	00'0104h	41h
See *Section 5.7*	XP2IR	XP2IE	XP2INT	00'0108h	42h
See *Section 5.7*	XP3IR	XP3IE	XP3INT	00'010Ch	43h

Table 16. 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	MAXIMAL 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 MAC Interruption Protected Instruction Fault Illegal Word Operand Access Illegal Instruction Access Illegal External Bus Access	 UNDOPC MACTRP PRTFLT ILLOPA ILLINA ILLBUS	 BTRAP BTRAP BTRAP BTRAP BTRAP BTRAP	 00'0028h 00'0028h 00'0028h 00'0028h 00'0028h 00'0028h	 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah	 I I I I I I minimal
Reserved			[2Ch –3Ch]	[0Bh – 0Fh]	
Software Traps TRAP Instruction			Any [00'0000h–00'01FCh] in steps of 4h	Any [00h – 7Fh]	Current CPU Priority

The *Table 16* 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 bits of an interrupt control register contain the complete interrupt status information of the associated source, which is required during one round of prioritization, the upper 8 bits of the respective register are reserved. All interrupt control registers are bit addressable and all bits can be read or written via software.

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

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

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

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

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

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

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

5.1.4 Interrupt priority level and group level

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

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

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

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

The interrupt system of the ST10F276 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.

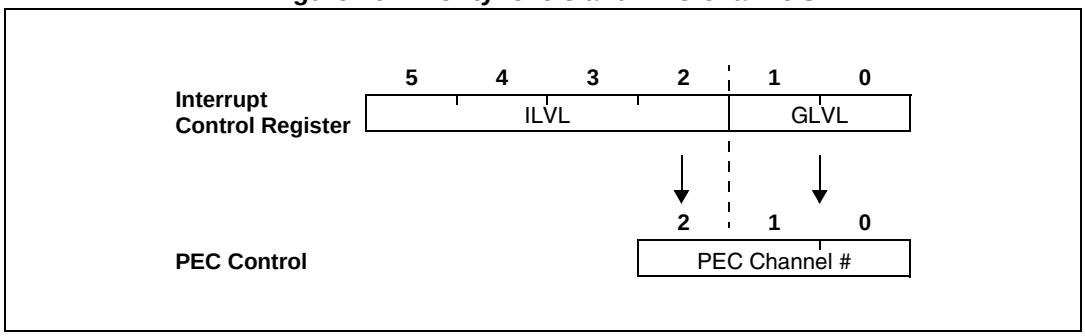
Note: Priority level 0000b is the default level of the CPU. Therefore a request on level 0 will never be serviced, because it can never interrupt the CPU. However, an enabled interrupt request on level 0000b will terminate the ST10F276's Idle mode and reactivate the CPU.

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 19*). 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 again *Figure 19*).

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 19. 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 two 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 ST10F276 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 Section 3: The central processing unit (CPU) on page 52).

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

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

Bit	Function
N, C, V, Z, E, MULIP, USR0	**CPU status flags** (Described in "*Section 3: The central processing unit (CPU) on page 52*") Define the current status of the CPU (ALU, Multiplication Unit).
HLDEN	**HOLD Enable** (Enables External Bus Arbitration) '0': Bus arbitration disabled, P6.7...P6.5 may be used for general purpose I/O. '1': Bus arbitration enabled, P6.7...P6.5 serve as BREQ, HLDA, HOLD, respectively.
IEN	**Interrupt Enable Control bit** (globally enables/disables interrupt requests) '0': Interrupt requests are disabled '1': Interrupt requests are enabled
ILVL	**CPU Priority Level** Defines the current priority level for the CPU. 'Fh': Highest priority level '0h': Lowest priority level

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 eight 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 (such as from serial channels, or A/D converter). Each channel is controlled by a dedicated PEC Channel Counter/Control register (PECCx) and a pair of pointers for source (SRCPx) and destination (DSTPx) of the data transfer. The PECC registers control the action that is performed by the respective PEC channel.

PECCx (FECyh / 6zh, see *Table 17*) SFR Reset Value: 0000h

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

Bit	Function
COUNT	**PEC Transfer Count** Counts PEC transfers and influences the channel's action (see table below).
BWT	**Byte / Word Transfer Selection** '0': Transfer a word '1': Transfer a byte
INC	**Increment Control** (Modification of SRCPx or DSTPx) '00': Pointers are not modified '01': Increment DSTPx by 1 or 2 (BWT) '10': Increment SRCPx by 1 or 2 (BWT) '11': Reserved. Do not use this combination (changed to '10' by hardware).

Table 17. 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.

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

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

Previous COUNT	Modified COUNT	IR after PEC service	Action of PEC channel and comments
FFh	FFh	'0'	Move a byte / word Continuous transfer mode, COUNT is not modified
FEh..02h	FDh..01h	'0'	Move a byte / word and decrement COUNT
01h	00h	'1'	Move a byte / word Leave request flag set, which triggers another request
00h	00h	('1')	**No action!** Activate interrupt service routine rather than PEC channel.

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

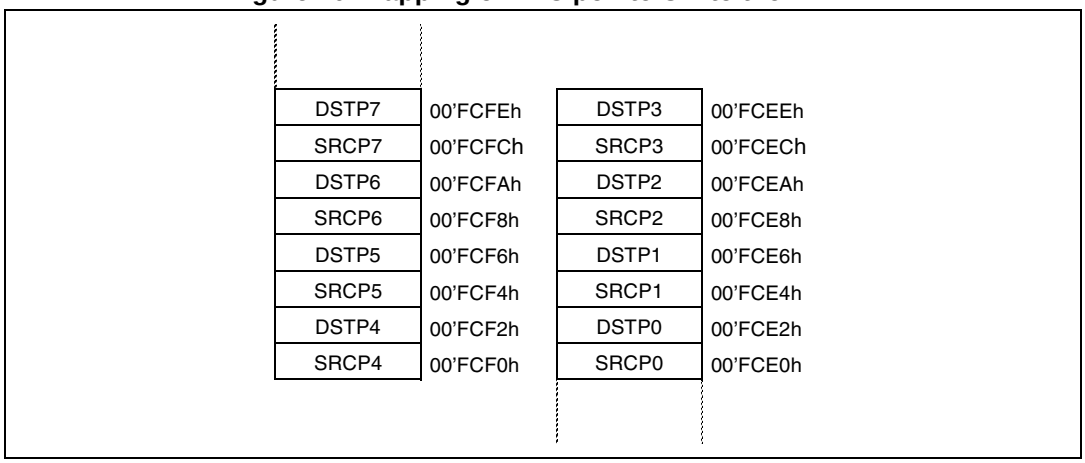
Continuous transfers are selected by the value FFh in bit-field COUNT. In this case COUNT is not modified and the respective PEC channel services any request until it is disabled again.

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

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

The source and destination pointers specify the locations between which the data is to be moved. A pair of pointers (SRCPx and DSTPx) is associated with each of the eight PEC channels. These pointers do not reside in specific SFRs, but are mapped into the IRAM of the ST10F276 just below the bit-addressable area (see *Figure 20*).

Figure 20. Mapping of PEC pointers into the IRAM



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 bits** allow to switch each individual source "ON" or "OFF", so it may generate a request or not. The control bits (xxIE) are located in the respective interrupt control registers. All interrupt requests may be enabled or disabled generally via bit IEN in register PSW. This control bit is the "main switch" that selects if requests from any source are accepted or not.
  In order to be arbitrated, both dedicated and global enabled bits of the interrupt source must be set.
- **The Priority Level** automatically selects a certain group of interrupt requests that will be acknowledged, disclosing all other requests. The priority level of the source that wins the arbitration is compared against the CPU's current level and only this source is

serviced. If its level is higher than the current CPU level. Changing the CPU level to a specific value via software blocks all requests on the same or a lower level. An interrupt source that is assigned to level 0 will be disabled and never be serviced.

- **The ATOMIC and EXTend instructions** automatically disable all interrupt requests for the duration of the following 1...4 instructions. This is useful for semaphore handling and does not require to re-enable the interrupt system after the inseparable instruction sequence (see *Section 27: System programming on page 545*).

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 ST10F276 supports this function with two features:

- Classes with up to four 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 four 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 three interrupt classes which cover two or three 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 18. Example of 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			

Table 18. Example of software controlled interrupt classes

ILVL (priority)	GLVL				Interpretation
	3	2	1	0	
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

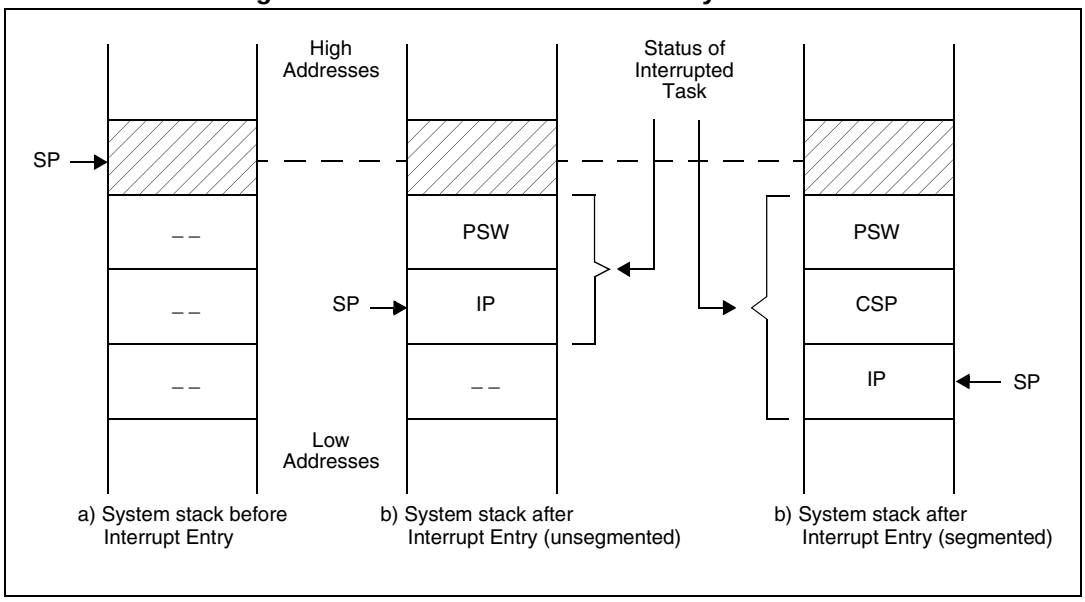
Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved along with the location where the execution of the interrupted task is to be resumed after returning from the service routine.

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 controls how the return location is stored.

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

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

Figure 21. 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 ST10F276 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

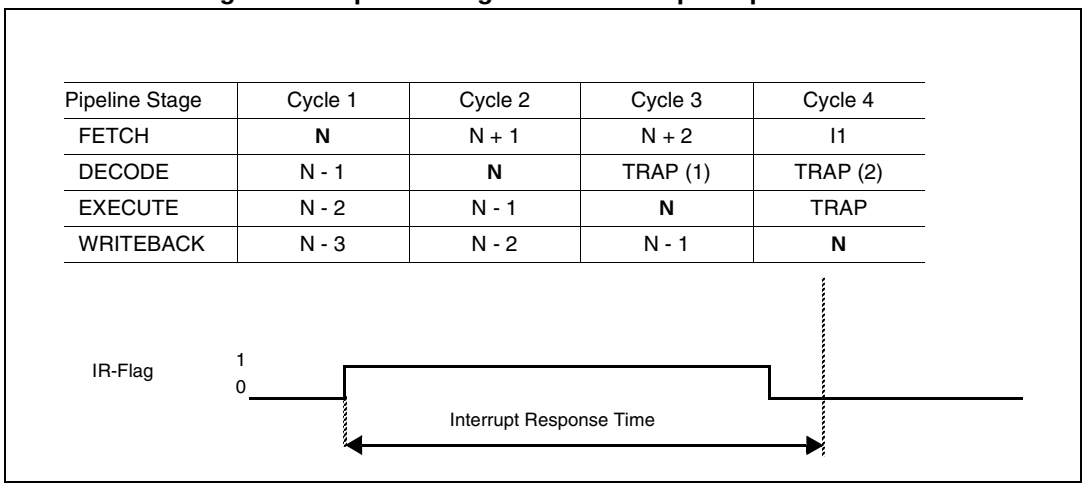
The interrupt response time defines the time from an interrupt request flag of an enabled interrupt source being set until the first instruction (I1) being fetched from the interrupt vector location. The basic interrupt response time for the ST10F276 is three instruction cycles (see *Figure 22*).

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

In the *Figure 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 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 22. Pipeline diagram for interrupt response time



The minimum interrupt response time is five CPU clock cycles. This requires program execution from the internal Flash, no external operand read requests and setting the interrupt request flag during the last CPU clock cycle of an instruction. When the interrupt request flag is set during the first CPU clock cycle of an instruction, the minimum interrupt response time is six 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 one 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 two CPU clock cycles during internal Flash program execution.

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

The worst case interrupt response time during internal Flash program execution adds to 12 CPU clock cycles.

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

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

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

The following is four examples to 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 nine word bus accesses, because instruction I1 cannot be fetched via the external bus until all write, fetch and read requests of preceding instructions in the pipeline are terminated.
- When the above example has the interrupt vector pointing into the internal Flash, the interrupt response time is 7 word bus accesses plus 2 CPU clock cycles, because fetching of instruction I1 from internal Flash can start earlier.
- When instructions N, N+1 and N+2 are executed out of external memory and the interrupt vector also points to an external location, but all operands for instructions N-3 through N are in internal memory, then the interrupt response time is the time to perform three word bus accesses.
- When the above example has the interrupt vector pointing into the internal Flash, the interrupt response time is one word bus access plus four CPU clock cycles.

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

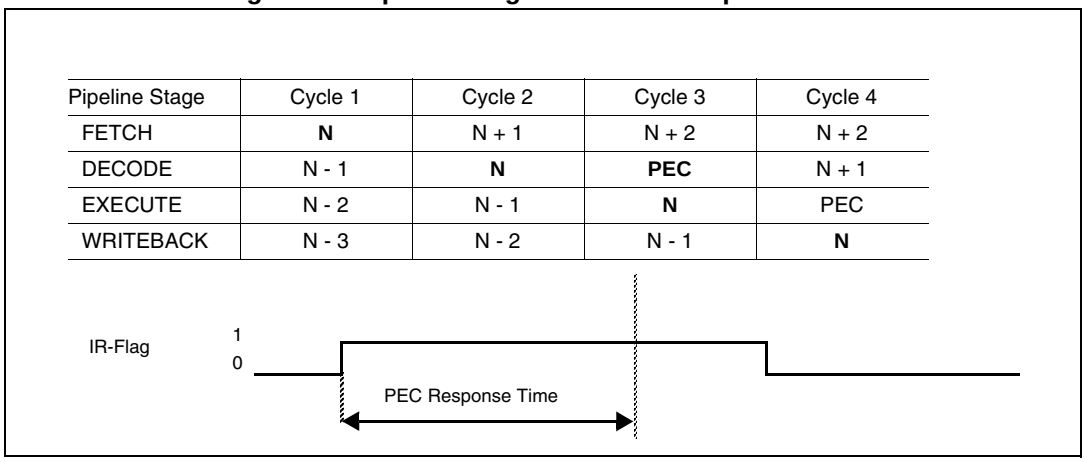
In most cases two instructions will be executed during this time. Only one instruction will typically be executed if the first instruction following the RETI instruction is a branch instruction (without cache hit), or if it reads an operand from internal Flash, or if it is executed out of the IRAM.

Note: A bus access in this context also includes delays caused by an external $\overline{\text{READY}}$ 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 ST10F276 is two instruction cycles.

Figure 23. Pipeline diagram for PEC response time



In *Figure 23* 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 three CPU clock cycles. This requires program execution from the internal Flash, no external operand read requests and setting the interrupt request flag during the last CPU clock cycle of an instruction. When the interrupt request flag is set during the first CPU clock cycle of an instruction, the minimum PEC response time is four CPU clock cycles. The PEC response time is increased by all delays of the instructions in the pipeline that are executed before starting the data transfer (including N):

- When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, the minimum PEC response time may be extended by 1 CPU clock cycle for each of these conditions.
- When instruction N reads an operand from the internal Flash, or when N is a CALL, RETURN, TRAP, or MOV Rn, [Rm+ #data16] instruction, the minimum PEC response time may additionally be extended by two CPU clock cycles during internal Flash program execution.
- In case instruction N reads the PSW and instruction N-1 has an effect on the condition flags, the PEC response time may additionally be extended by two CPU clock cycles.

The worst case PEC response time during internal Flash program execution adds to nine 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 is a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case PEC 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 seven 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 two CPU clock cycles.

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

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

5.6 External interrupts

Although the ST10F276 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 19 on page 114).

When port pins CCxIO 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 CCxIO.

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 19*).

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 19. Pins to be used as external interrupt inputs

Port pin	Original function	Control register
P2.0-15/CC0-15IO	CAPCOM Register 0-15 Capture Input	CC0-CC15
P8.0-7/CC16-23IO	CAPCOM Register 16-23 Capture Input	CC16-CC23
P1H.4-7/CC24-27I	CAPCOM Register 24-27 Capture Input	CC24-CC27
P7.4-7/CC28-31IO	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 pin $\overline{RSTIN}$ provide another possibility for the CPU to react on an external input signal. NMI and 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 eight CPU clock cycles: This means that the external events are scanned and detected in timeframes of eight CPU clock cycles.

The ST10F276 provides eight interrupt inputs that are sampled every CPU clock cycle so external events are captured faster than with standard interrupt inputs.

The upper eight pins of Port2 (CC8-15IO 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 eight pins. In addition, these fast interrupt inputs feature programmable edge detection (rising edge, falling edge or both edges).

Fast external interrupts may also have interrupt sources selected from other peripherals; for example the CANx controller receive signal (CANx_RxD) can be used to interrupt the system. The same is valid for I^2C interface serial clock line (alternative to CAN2 receive line), and for Real Time Clock (internally managed). This function is controlled using the 'External Interrupt Source Selection' register EXISEL.

The EXxIN pins (P2.8-P2.15) can also be used to exit Power Down mode if bit PWDCFG in the SYSCON register is set. Power reduction modes are detailed in *Section 24: Power reduction modes on page 502*.

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

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

Bit	Function
EXIxES (x=7...0)	**External Interrupt x Edge Selection Field** (x=3...0) '00': Fast external interrupts disabled: standard mode. EXxIN pin not taken into account for entering/exiting Power Down mode. '01': Interrupt on positive edge (rising). Enter Power Down mode if EXxIN = '0', exit if EXxIN = '1' (ref as 'high' active level). '10': Interrupt on negative edge (falling). Enter Power Down mode if EXxIN = '1', exit if EXxIN = '0' (ref as 'low' active level). '11': Interrupt on any edge (rising or falling). Always enter Power Down mode, exit if EXxIN level changed.

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

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

While for CAN and I^2C the EXICON programming depends on the customer application (even though inactive state of both CAN and I^2C protocols is the high level, so a new activity on the bus can be detected by a falling edge observed at the related pins), for RTC the internal hardware circuitry is such that the interrupts are generated on the positive edge, so the EXICON register must be programmed accordingly.

Note: The I²C interface implements an input analog filter to avoid that spurious spikes are taken into account as valid bus transitions. For this reason, a pulse on the SCL line should be long enough to be recognized as valid pulse: this is in the range of 500ns (minimum). All pulses shorter than 50ns are certainly filtered: a pulse longer than 50ns but shorter than 500ns could either trigger or not trigger the exit from Power Down mode.

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

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

EXIxSS	Port2 pin	Alternate source	
0	P2.8	CAN1_RxD	P4.5
1	P2.9	CAN2_RxD / SCL	P4.4
2	P2.10	RTCSI (Second)	Internal MUX
3	P2.11	RTCAI (Alarm)	Internal MUX
4...7	P2.12...15	Not used (zero)	-

Bit	Function
EXIxSS	**External Interrupt x Source Selection** (x=7...0) ‘00’: Input from associated Port2 pin. ‘01’: Input from “alternate source”. ‘10’: Input from Port2 pin ORed with “alternate source”. ‘11’: Input from Port2 pin ANDed with “alternate source”.

CAN and I²C interrupt mapping need some considerations to exploit all possible configurations of the pin mapping and function enabling. In particular, when a module is not enabled, even though the interrupt source is enabled (see for example EXIxSS=’01’) an event on the pin does not generate any request to the CPU.

In the next table, all the possible pin configurations are summarized (considering I²C and CAN2 pin sharing and CAN parallel mode). In the table, the bits of XPERCON register (used to enable/disable each module) and the bit CANPAR of XMISC register (used to enable/disable the CAN parallel mode) are reported. The table indicates when the interrupt (or Power Down exiting) can be generated by the three modules (supposing to have properly set register EXISEL).

The two general rules are the following:

- CAN parallel mode is enabled only when both CAN modules are enabled (if not, it has no effect).
- When I²C is enabled, CAN2 enabling has no effect.

CANPAR	XI2CEN	CAN2EN	CAN1EN	Interrupt P4.5	Interrupt P4.4
x	0	0	0	No	No
x	0	0	1	Yes (CAN1)	No
x	0	1	0	No	Yes (CAN2)
0	0	1	1	Yes (CAN1)	Yes (CAN2)
1	0	1	1	Yes (CAN1/2)	No
x	1	0	0	No	Yes (I^2C)
x	1	0	1	Yes (CAN1)	Yes (I^2C)
x	1	1	0	No	Yes (I^2C)
0	1	1	1	Yes (CAN1)	Yes (I^2C)
1	1	1	1	Yes (CAN1/2)	Yes (I^2C)

EXxIN inputs are normally sampled interrupt inputs. However, the interrupt handler circuitry uses them as level-sensitive inputs. An EXxIN (x = 7...0) Interrupt Enable bit (bit CCxIE in respective CCxIC register) needs not to be set to properly serve the interrupt request (or exiting from Power Down mode).

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. 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: For CAN1 (and CAN2 when parallel mode is set) the related interrupt control register is CC8IC; for CAN2 and I^2C the register is CC9IC.

5.7 X-peripheral interrupt

The limited number of X-Bus interrupt lines of the present ST10 architecture, imposes some constraints on the implementation of the new functionality. In particular, the additional X-Peripherals XSSC, XASC, I^2C, XPWM and RTC need some resources to implement interrupt and PEC transfer capabilities. For this reason, a sophisticated but very flexible multiplexed structure for the interrupt management is proposed. In *Figure 24*, the principle is explained through a simple diagram, which shows the basic structure replicated for each of the four X-interrupt available vectors (XP0INT, XP1INT, XP2INT and XP3INT).

It is based on a set of 16-bit registers XIRxSEL (x=0,1,2,3), divided in two portions each:

- Byte HighXIRxSEL[15:8]Interrupt Enable bits
- Byte LowXIRxSEL[7:0]Interrupt Flag bits

When different sources submit an interrupt request, the enable bits (Byte High of XIRxSEL register) define a mask which controls which sources will be associated with the unique available vector. If more than one source is enabled to issue the request, the service routine will have to take care to identify the real event to be serviced. This can easily be done by checking the flag bits (Byte Low of XIRxSEL register). Note that the flag bits can also provide information about events which are not currently serviced by the interrupt controller (since masked through the enable bits), allowing an effective software management also in

absence of the possibility to serve the related interrupt request: a periodic polling of the flag bits may be implemented inside the user application.

Note: The XIRxSEL registers are mapped into the XMiscellaneous area. Therefore they can be accessed only if the XMISCEN bit is set in XPERCON register and if XPEN bit is set in SYSCON register.

Figure 24. X-Interrupt basic structure

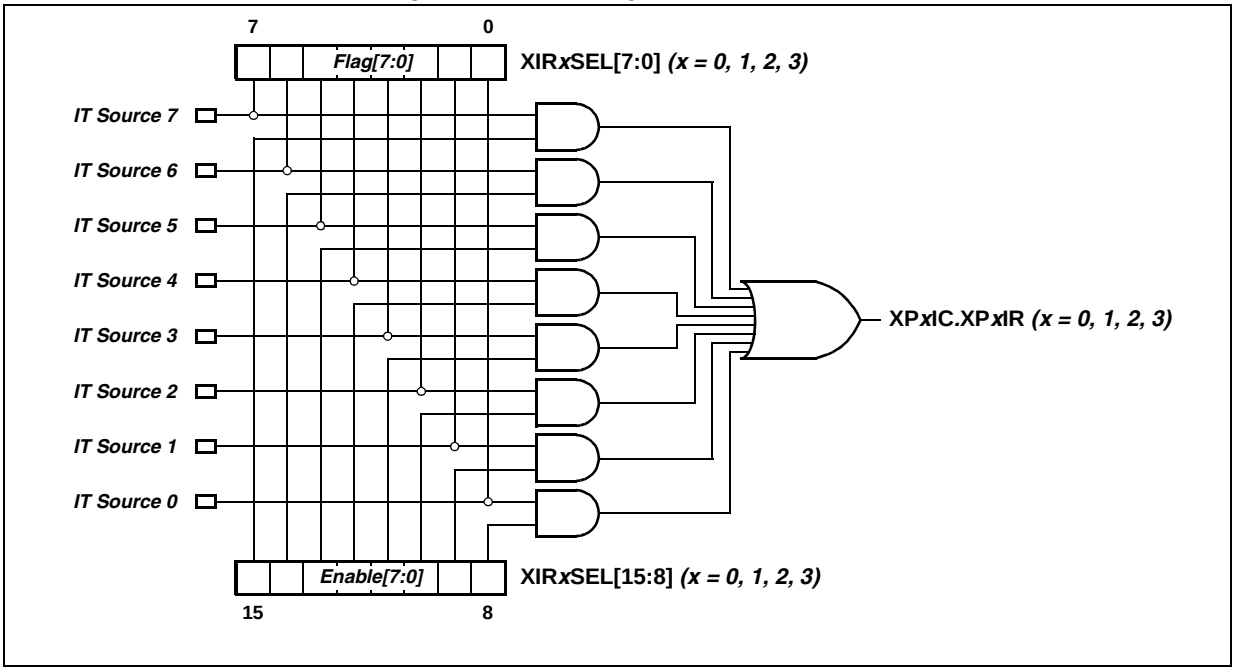


Table 20 summarizes the mapping of the different interrupt sources which shares the four X-interrupt vectors. For details on bits inside the XIRxSEL registers, refer to register description section reported just after the table itself.

Since the XIRxSEL registers are not bit addressable, another pair of registers (a couple for each XIRxSEL) is provided to allow setting and clearing the bits of XIRxSEL without risking to overwrite requests coming after reading the register and before writing it. These registers are described in this section as well.

Table 20. X-Interrupt detailed mapping

	XP0INT	XP1INT	XP2INT	XP3INT
CAN1 Interrupt	x			x
CAN2 Interrupt		x		x
I^2C Receive	x	x	x	
I^2C Transmit	x	x	x	
I^2C Error				x
XSSC Receive	x	x	x	
XSSC Transmit	x	x	x	
XSSC Error				x

Table 20. X-Interrupt detailed mapping

	XP0INT	XP1INT	XP2INT	XP3INT
XASC Receive	x	x	x	
XASC Transmit	x	x	x	
XASC Transmit Buffer	x	x	x	
XASC Error				x
PLL Unlock / OWD				x
PWM1 Channel 3...0			x	x

XIR0SEL (EB10h) XBUS Reset Value: 0000h

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

Bit	Function
FL.0	**Interrupt Flag 0: CAN1 Interrupt** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.1	**Interrupt Flag 1: I^2C Transmit** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.2	**Interrupt Flag 2: I^2C Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.3	**Interrupt Flag 0: XSSC Transmit** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.4	**Interrupt Flag 4: XSSC Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.5	**Interrupt Flag 5: XASC Transmit Buffer** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.6	**Interrupt Flag 6: XASC Transmit** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.7	**Interrupt Flag 7: XASC Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
IE.0	**Interrupt Enable 0: CAN1 Interrupt** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.

Bit	Function
IE.1	**Interrupt Enable 1: I2C Transmit** '0': Interrupt request disabled. '1': Interrupt request enabled.
IE.2	**Interrupt Enable 2: I2C Receive** '0': Interrupt request disabled. '1': Interrupt request enabled.
IE.3	**Interrupt Enable 3: XSSC Transmit** '0': Interrupt request disabled. '1': Interrupt request enabled.
IE.4	**Interrupt Enable 4: XSSC Receive** '0': Interrupt request disabled. '1': Interrupt request enabled.
IE.5	**Interrupt Enable 5: XASC Transmit Buffer** '0': Interrupt request disabled. '1': Interrupt request enabled.
IE.6	**Interrupt Enable 6: XASC Transmit** '0': Interrupt request disabled. '1': Interrupt request enabled.
IE.7	**Interrupt Enable 7: XASC Receive** '0': Interrupt request disabled. '1': Interrupt request enabled.

All bits of XIR0SEL register are set by hardware when an interrupt is coming from the peripheral, and/or by writing a logic '1' in the corresponding bit of XIR0SET register.

All bits of XIR0SEL register are cleared by writing a logic '1' in the corresponding bit of XIR0CLR register. The register can anyway be also written directly by the software.

XIR0SET (EB12h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IESET[7:0]								FLSET[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLSET.x	**Interrupt Flag x SET** (x=7...0) Writing a '1' will set the corresponding bit *x* in XIR0SEL register. Writing a '0' has no effect.
IESET.x	**Interrupt Enable x SET** (x=7...0) Writing a '1' will set the corresponding bit *x* in XIR0SEL register. Writing a '0' has no effect.

XIR0CLR (EB14h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IECLR[7:0]								FLCLR[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLCLR.x	**Interrupt Flag x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR0SEL register. Writing a '0' has no effect.
IECLR.x	**Interrupt Enable x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR0SEL register. Writing a '0' has no effect.

To enable the interrupt in the interrupt controller, the Interrupt Control Register XP0IC has to be initialized. The associated interrupt vector is called XP0INT located at address 100h (trap number 40h).

XP0IC (F186h / C3h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	XP0IR	XP0IE	ILVL				GLVL	
								RW	RW	RW				RW	

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

XIR1SEL (EB20h) XBUS Reset Value: 0000h

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

Bit	Function
FL.0	**Interrupt Flag 0: CAN2 Interrupt** '0': No interrupt request. '1': Interrupt request pending.
FL.1	**Interrupt Flag 1: I2C Transmit** '0': No interrupt request. '1': Interrupt request pending.
FL.2	**Interrupt Flag 2: I2C Receive** '0': No interrupt request. '1': Interrupt request pending.
FL.3	**Interrupt Flag 0: XSSC Transmit** '0': No interrupt request. '1': Interrupt request pending.

Bit	Function
FL.4	**Interrupt Flag 4: XSSC Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.5	**Interrupt Flag 5: XASC Transmit Buffer** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.6	**Interrupt Flag 6: XASC Transmit** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.7	**Interrupt Flag 7: XASC Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
IE.0	**Interrupt Enable 0: CAN2 Interrupt** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.1	**Interrupt Enable 1: I2C Transmit** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.2	**Interrupt Enable 2: I2C Receive** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.3	**Interrupt Enable 3: XSSC Transmit** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.4	**Interrupt Enable 4: XSSC Receive** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.5	**Interrupt Enable 5: XASC Transmit Buffer** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.6	**Interrupt Enable 6: XASC Transmit** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.7	**Interrupt Enable 7: XASC Receive** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.

All bits of XIR1SEL register are set by hardware when an interrupt is coming from the peripheral, and/or by writing a logic ‘1’ in the corresponding bit of XIR1SET register. All bits of XIR1SEL register are cleared by writing a logic ‘1’ in the corresponding bit of XIR1CLR register. The register can anyway be also written directly by the software.

XIR1SET (EB22h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IESET[7:0]								FLSET[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLSET.x	**Interrupt Flag x SET** (x=7...0) Writing a '1' will set the corresponding bit *x* in XIR1SEL register. Writing a '0' has no effect.
IESET.x	**Interrupt Enable x SET** (x=7...0) Writing a '1' will set the corresponding bit *x* in XIR1SEL register. Writing a '0' has no effect.

XIR1CLR (EB24h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IECLR[7:0]								FLCLR[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLCLR.x	**Interrupt Flag x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR1SEL register. Writing a '0' has no effect.
IECLR.x	**Interrupt Enable x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR1SEL register. Writing a '0' has no effect.

To enable the interrupt in the interrupt controller, the Interrupt Control Register XP1IC has to be initialized. The associated interrupt vector is called XP1INT located at address 104h (trap number 41h).

XP1IC (F18Eh / C7h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	XP1IR	XP1IE	ILVL				GLVL	
								RW	RW	RW				RW	

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

XIR2SEL (EB30h) XBUS Reset Value: 0000h

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

Bit	Function
FL.0	**Interrupt Flag 0: XPWM Channel 3...0** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.1	**Interrupt Flag 1: I2C Transmit** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.2	**Interrupt Flag 2: I2C Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.3	**Interrupt Flag 0: XSSC Transmit** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.4	**Interrupt Flag 4: XSSC Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.5	**Interrupt Flag 5: XASC Transmit Buffer** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.6	**Interrupt Flag 6: XASC Transmit** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.7	**Interrupt Flag 7: XASC Receive** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
IE.0	**Interrupt Enable 0: XPWM Channel 3...0** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.1	**Interrupt Enable 1: I2C Transmit** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.2	**Interrupt Enable 2: I2C Receive** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.3	**Interrupt Enable 3: XSSC Transmit** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.4	**Interrupt Enable 4: XSSC Receive** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.5	**Interrupt Enable 5: XASC Transmit Buffer** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.

Bit	Function
IE.6	**Interrupt Enable 6: XASC Transmit** '0': Interrupt request disabled. '1': Interrupt request enabled.
IE.7	**Interrupt Enable 7: XASC Receive** '0': Interrupt request disabled. '1': Interrupt request enabled.

All bits of XIR2SEL register are set by hardware when an interrupt is coming from the peripheral, and/or by writing a logic '1' in the corresponding bit of XIR2SET register. All bits of XIR2SEL register are cleared by writing a logic '1' in the corresponding bit of XIR2CLR register. The register can anyway be also written directly by the software.

XIR2SET (EB32h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IESET[7:0]								FLSET[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLSET.x	**Interrupt Flag x SET** (x=7...0) Writing a '1' will set the corresponding bit *x* in XIR2SEL register. Writing a '0' has no effect.
IESET.x	**Interrupt Enable x SET** (x=7...0) Writing a '1' will set the corresponding bit *x* in XIR2SEL register. Writing a '0' has no effect.

XIR2CLR (EB34h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IECLR[7:0]								FLCLR[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLCLR.x	**Interrupt Flag x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR2SEL register. Writing a '0' has no effect.
IECLR.x	**Interrupt Enable x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR2SEL register. Writing a '0' has no effect.

To enable the interrupt in the interrupt controller, the Interrupt Control Register XP2IC has to be initialized. The associated interrupt vector is called XP2INT located at address 108h (trap number 42h).

XP2IC (F196h / CBh) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	XP2IR	XP2IE	ILVL				GLVL	
								RW	RW	RW				RW	

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

XIR3SEL (EB40h) XBUS Reset Value: 0000h

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

Bit	Function
FL.0	**Interrupt Flag 0: CAN1 Interrupt** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.1	**Interrupt Flag 1: CAN2 Interrupt** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.2	**Interrupt Flag 2: I2C Error** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.3	**Interrupt Flag 0: XSSC Error** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.4	**Interrupt Flag 4: XASC Error** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.5	**Interrupt Flag 5: PLL Unlock / Oscillator Watchdog** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.6	**Interrupt Flag 6: XPWM Channel 3...0** ‘0’: No interrupt request. ‘1’: Interrupt request pending.
FL.7	**Interrupt Flag 7:** *No interrupt source associated.*
IE.0	**Interrupt Enable 0: CAN1 Interrupt** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.1	**Interrupt Enable 1: CAN2 Interrupt** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.2	**Interrupt Enable 2: I2C Error** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.

Bit	Function
IE.3	**Interrupt Enable 3: XSSC Error** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.4	**Interrupt Enable 4: XASC Error** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.5	**Interrupt Enable 5: PLL Unlock / Oscillator Watchdog** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.6	**Interrupt Enable 6: XPWM Channel 3...0** ‘0’: Interrupt request disabled. ‘1’: Interrupt request enabled.
IE.7	**Interrupt Enable 7:** *No interrupt source associated.*

All bits of XIR3SEL register are set by hardware when an interrupt is coming from the peripheral, and/or by writing a logic ‘1’ in the corresponding bit of XIR3SET register. All bits of XIR3SEL register are cleared by writing a logic ‘1’ in the corresponding bit of XIR3CLR register. The register can anyway be also written directly by the software.

XIR3SET (EB42h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IESET[7:0]								FLSET[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLSET.x	**Interrupt Flag x SET** (x=7...0) Writing a ‘1’ will set the corresponding bit *x* in XIR3SEL register. Writing a ‘0’ has no effect.
IESET.x	**Interrupt Enable x SET** (x=7...0) Writing a ‘1’ will set the corresponding bit *x* in XIR3SEL register. Writing a ‘0’ has no effect.

XIR3CLR (EB44h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IECLR[7:0]								FLCLR[7:0]							
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
FLCLR.x	**Interrupt Flag x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR3SEL register. Writing a '0' has no effect.
IECLR.x	**Interrupt Enable x CLEAR** (x=7...0) Writing a '1' will clear the corresponding bit *x* in XIR3SEL register. Writing a '0' has no effect.

To enable the interrupt in the interrupt controller, the Interrupt Control Register XP3IC has to be initialized. The associated interrupt vector is called XP3INT located at address 10Ch (trap number 43h).

XP3IC (F19Eh / CFh) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	XP3IR	XP3IE	ILVL				GLVL	
								RW	RW	RW				RW	

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

5.8 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 ST10F276 provides two different kinds of trap mechanisms:

- **Hardware traps** are triggered by events that occur during program execution (like illegal access or undefined opcode);
- **Software traps** are initiated via an instruction within the current execution flow.

The trap priorities are summarized in *Table 21*.

Table 21. Trap priorities

Exception condition	Trap flag	Trap vector	Vector location	Trap number	Trap priority
RESET Functions:					MAXIMAL
Hardware RESET		RESET	00'0000h	00h	III
Software RESET		RESET	00'0000h	00h	III
Watchdog Timer Overflow		RESET	00'0000h	00h	III
Class A Hardware Traps:					
Non-Maskable Interrupt	NMI	NMITRAP	00'0008h	02h	II
Stack Overflow	STKOF	STOTRAP	00'0010h	04h	II
Stack Underflow	STKUF	STUTRAP	00'0018h	06h	II

Table 21. Trap priorities

Exception condition	Trap flag	Trap vector	Vector location	Trap number	Trap priority
Class B Hardware Traps: Undefined Opcode MAC Interruption Protected Instruction Fault Illegal Word Operand Access Illegal Instruction Access Illegal External Bus Access	 UNDOPC MACTRP PRTFLT ILLOPA ILLINA ILLBUS	 BTRAP BTRAP BTRAP BTRAP BTRAP BTRAP	 00'0028h 00'0028h 00'0028h 00'0028h 00'0028h 00'0028h	 0Ah 0Ah 0Ah 0Ah 0Ah 0Ah	 I I I I I I MINIMAL
Reserved			[2Ch –3Ch]	[0Bh – 0Fh]	
Software Traps TRAP Instruction			Any [00'0000h– 00'01FCh] in steps of 4h	Any [00h – 7Fh]	Current CPU Priority

5.8.1 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.

5.8.2 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 ST10F276 distinguishes nine 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 *Table 5.1: Interrupt system structure on page 96*).

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 nine hardware trap functions of the ST10F276 are divided into two classes:

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

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

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

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

Bit	Function
ILLBUS	**Illegal External Bus Access Flag** An external access has been attempted with no external bus defined.
ILLINA	**Illegal Instruction Access Flag** A branch to an odd address has been attempted.
ILLOPA	**Illegal Word Operand Access Flag** A word operand access (read or write) to an odd address has been attempted.
PRTFLT	**Protection Fault Flag** A protected instruction with an illegal format has been detected.
MACTRP	**MAC Interrupt Flag** The MAC co-processor has generated an interruption.
UNDOPC	**Undefined Opcode Flag** The currently decoded instruction has no valid ST10F276 opcode.

Bit	Function
STKUF	**Stack Underflow Flag** The current stack pointer value exceeds the content of register STKUN.
STKOF	**Stack Overflow Flag** The current stack pointer value falls below the content of register STKOV.
NMI	**Non Maskable Interrupt Flag** A negative transition (falling edge) has been detected on pin $\overline{\text{NMI}}$.

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

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

Class A traps have the second highest priority (Trap Priority II), on the third rank are class B traps, so a class A trap can interrupt a class B trap. If more than one class A trap occur at a time, they are prioritized internally, with the NMI trap on the highest priority followed by the stack overflow trap; the stack underflow trap has the lowest priority.

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

If a class A trap occurs during the execution of a class B trap service routine, class A trap will be serviced immediately. During the execution of a class A trap service routine, no class B trap will be serviced until the class A trap service routine is exited with a RETI instruction. In this case, the occurrence of the class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap is lost.

If an Undefined Opcode trap (class B) occurs simultaneously with an NMI trap (class A), both the NMI and the UNDOPC flags are 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.

5.8.3 External NMI trap

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

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

5.8.4 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.

5.8.5 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.

5.8.6 Undefined opcode trap

When the instruction currently decoded by the CPU does not contain a valid ST10F276 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.

5.8.7 MAC interrupt

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

5.8.8 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.

5.8.9 Illegal word operand access trap

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

5.8.10 Illegal instruction access trap

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

5.8.11 Illegal external bus access trap

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

6 Parallel ports

6.1 Introduction

The ST10F276 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, Port6, Port7, 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 ST10F276'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 six I/O ports (2, 3, partially 4, 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.

Note: A set of registers mapped on XBUS is implemented on ST10F276 to manage the data, direction and open-drain mode for those pins where the new X-Peripherals (XPWM, XASC and XSSC) are mapped: the standard Port register bits for these pins are working when the X-Peripherals are not enabled (see XPERCON register); vice versa, when the X-Peripherals are enabled, the new registers take the control of the pins, and the standard registers content is ignored.

6.1.1 Open drain mode

Some of I/O Ports of ST10F276 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, P4 (partially), P6, P7 and P8 (see respective sections), and is controlled through the respective Open Drain Control Registers ODPx.

These registers allow the individual bit wise selection of the open drain mode for each port line. If the respective control bit ODPx.y is '0' (default after reset), the output driver is in the push / pull mode. If ODPx.y is '1', the open drain configuration is selected. Note that all ODPx registers are located in the ESFR space (see *Figure 26*).

Note: When XPWM, XASC and XSSC are used (enabled through XPERCON) the open-drain mode of the related pins is controlled by a set of XBUS registers (XPWMPORT, XS1PORT, XSSCPORT).
When I^2C is enabled (through XPERCON), the related pins of Port4 are automatically set to open-drain mode (ODP4 register is bypassed when I^2C is active; besides CAN2 is not accessible when I^2C is active unless remapped in parallel to CAN1).

Figure 25. SFRs, XBUS registers and pins associated with the parallel ports

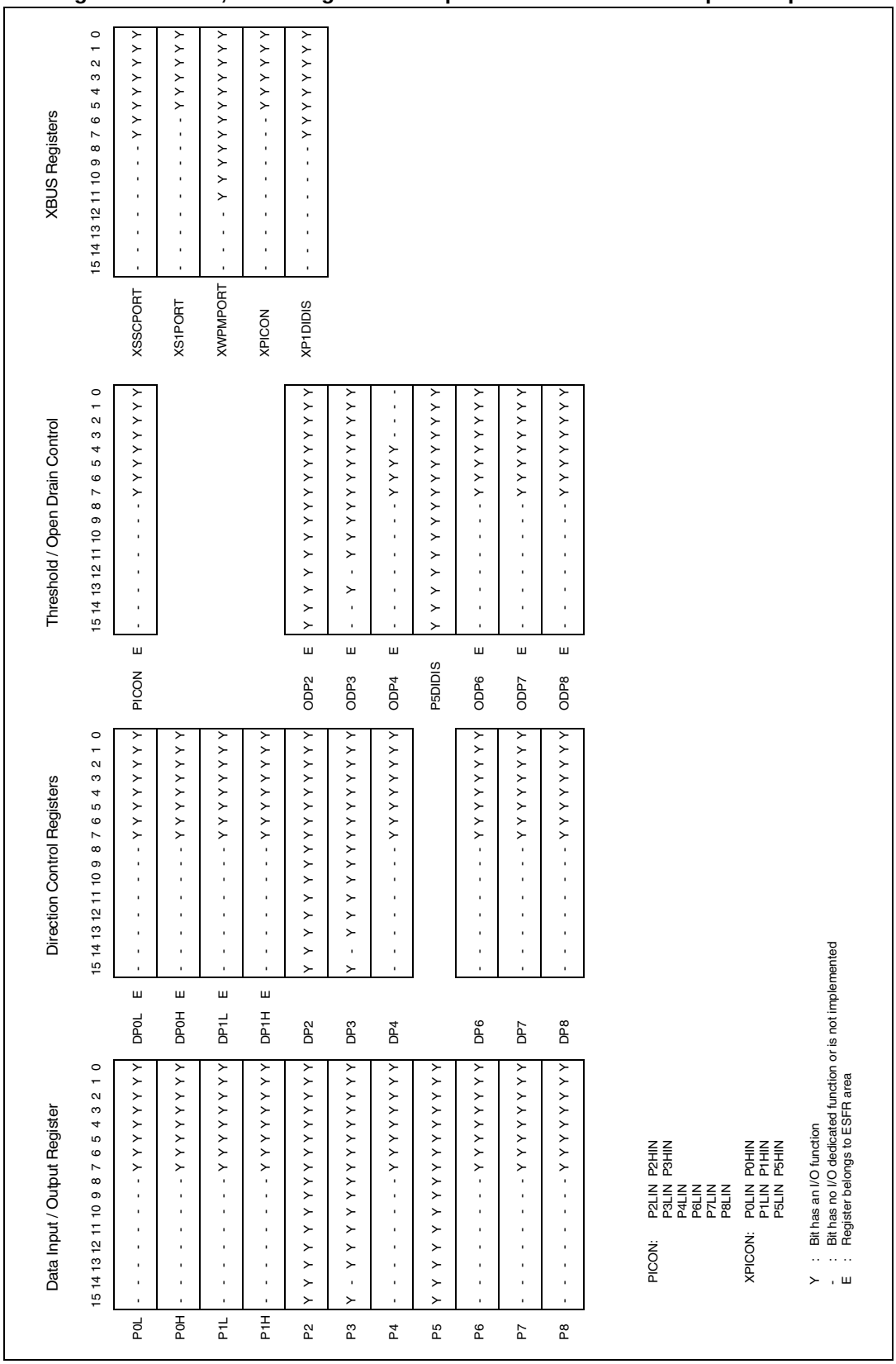
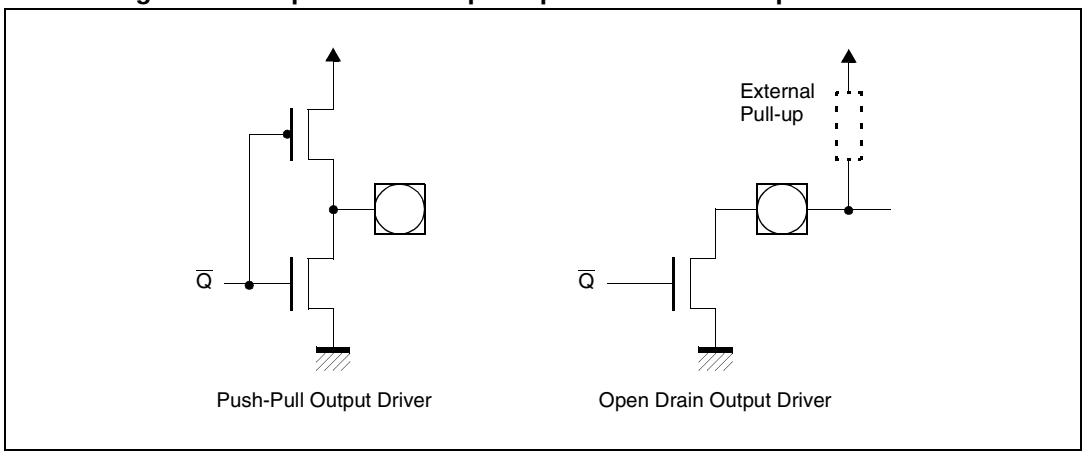


Figure 26. Output drivers in push-pull mode and in open drain mode



6.1.2 Input threshold control

The standard inputs of the ST10F276 determine the status of input signals according to TTL levels.

In order to accept and recognize noisy signals, CMOS input thresholds can be selected instead of the standard TTL thresholds for all pins of all Ports. These CMOS configuration features also a higher hysteresis, to prevent the inputs from toggling while the respective input signal level is near the thresholds.

Two Port Input Control registers (PICON and XPICON) are used to select these thresholds for each byte of the indicated ports: the 8-bit ports P4, P6, P7 and P8 are controlled by one bit each while ports P0, P1, P2, P3 and P5 are controlled by two bits each.

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	P8LIN	P7LIN	P6LIN	P4LIN	P3HIN	P3LIN	P2HIN	P2LIN
								RW	RW	RW	RW	RW	RW	RW	RW

XPICON (EB26h) XBUS Reset Value: - - 00h

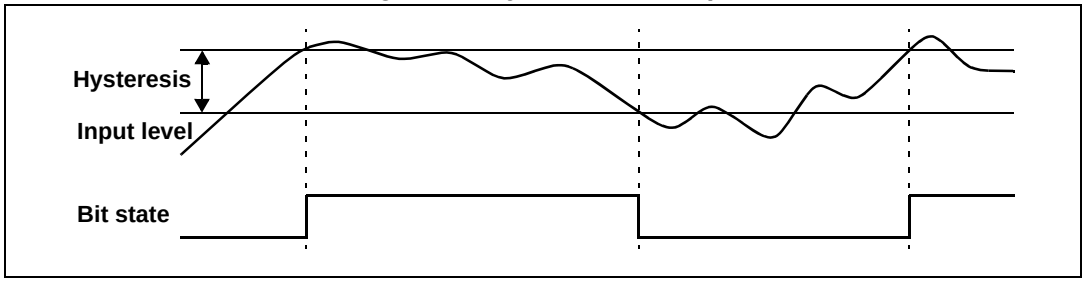
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	P5HIN	P5LIN	P1HIN	P1LIN	P0HIN	P0LIN
										RW	RW	RW	RW	RW	RW

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

Note: PICON is an ESFR register, while XPICON is an XBUS register. To access XPICON register bit XMISCEN of register XPERCON and bit XPEN of register SYSCON must be set.

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

Figure 27. Hysteresis concept



6.1.3 Alternate port functions

Each port line has one or more associated programmable alternate input or output function. PORT0 and PORT1 may be used as the address and data lines when accessing external memory. Besides, PORT1 provides also Input Capture lines and additional (8) analog input channels to the A/D Converter.

Port4 outputs the additional segment address bit A23...A16 in systems where more than 64 Kbytes of memory are to be accessed directly. In addition, CAN1, CAN2 and I^2C lines are provided.

Port6 provides the optional chip select outputs, the bus arbitration lines, and the XSSC 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, of the XPWM module and of the XASC.
Port2 is also used for fast external interrupt inputs and for timer 7 input.

Port3 includes alternate input/output functions of timers, standard serial interfaces (SSC and ASC), the optional bus control signal $\overline{BHE}/\overline{WRH}$ and the system clock output (CLKOUT). Port5 is used for the analog input channels (16) to the A/D converter or timer control signals.

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

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

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

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

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

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

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

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

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

All port lines that are not used for these alternate functions may be used as general purpose I/O lines. When using port pins for general purpose output, the initial output value should be written to the port latch prior to enabling the output drivers, in order to avoid undesired transitions on the output pins. This applies to single pins as well as to pin groups (see examples below).

Note: When using several BSET pairs to control more pins of one port, these pairs must be separated by instructions, which do not reference the respective port (see Section 3.1.4: Particular pipeline effects on page 56).

```
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
```

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 DP0H and DP0L.

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

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

P0H (FF02h / 81h) SFR Reset Value: - - 00h

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

Bit	Function
P0X.y	Port data register P0H or P0L bit y

DP0L (F100h / 80h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	DP0L.7	DP0L.6	DP0L.5	DP0L.4	DP0L.3	DP0L.2	DP0L.1	DP0L.0
								RW	RW	RW	RW	RW	RW	RW	RW

DP0H (F102h / 81h) ESFR Reset Value: - - 00h

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

Bit	Function
DP0X.y	**Port direction register DP0H or DP0L bit y** '0': Port line P0X.y is an input (high-impedance). '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 datasheets) 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 datasheet 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.

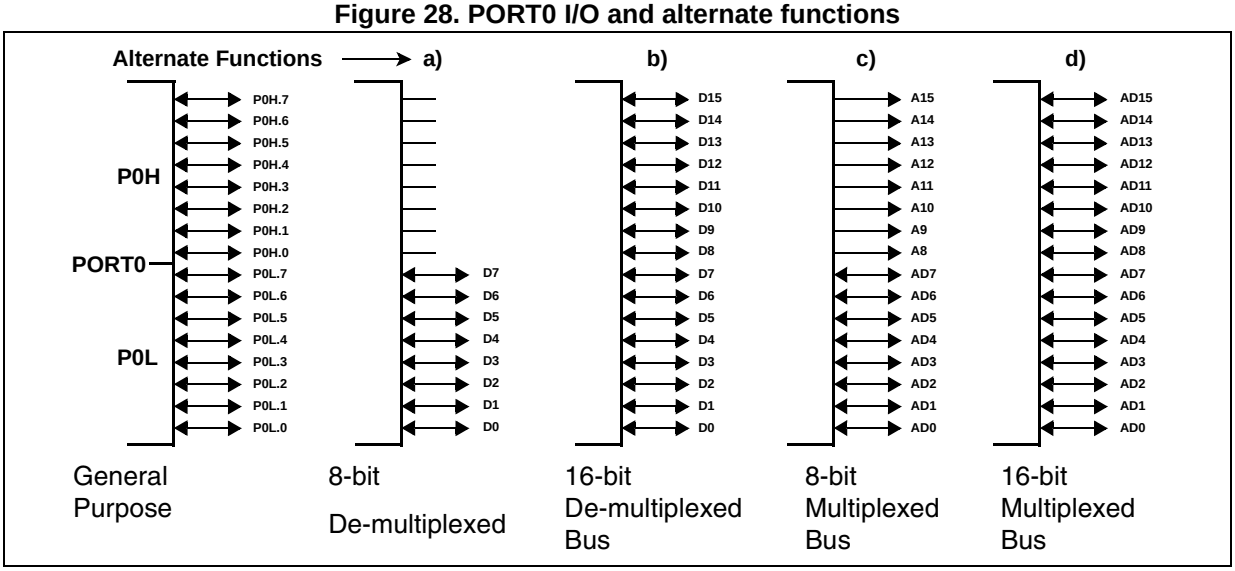
This read-only register holds the selection for the number of chip selects and segment addresses. Software can read this register in order to react according to the selected configuration, if required. When the reset is terminated, the internal pull-up devices are switched off, and PORT0 will be switched 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 28*).

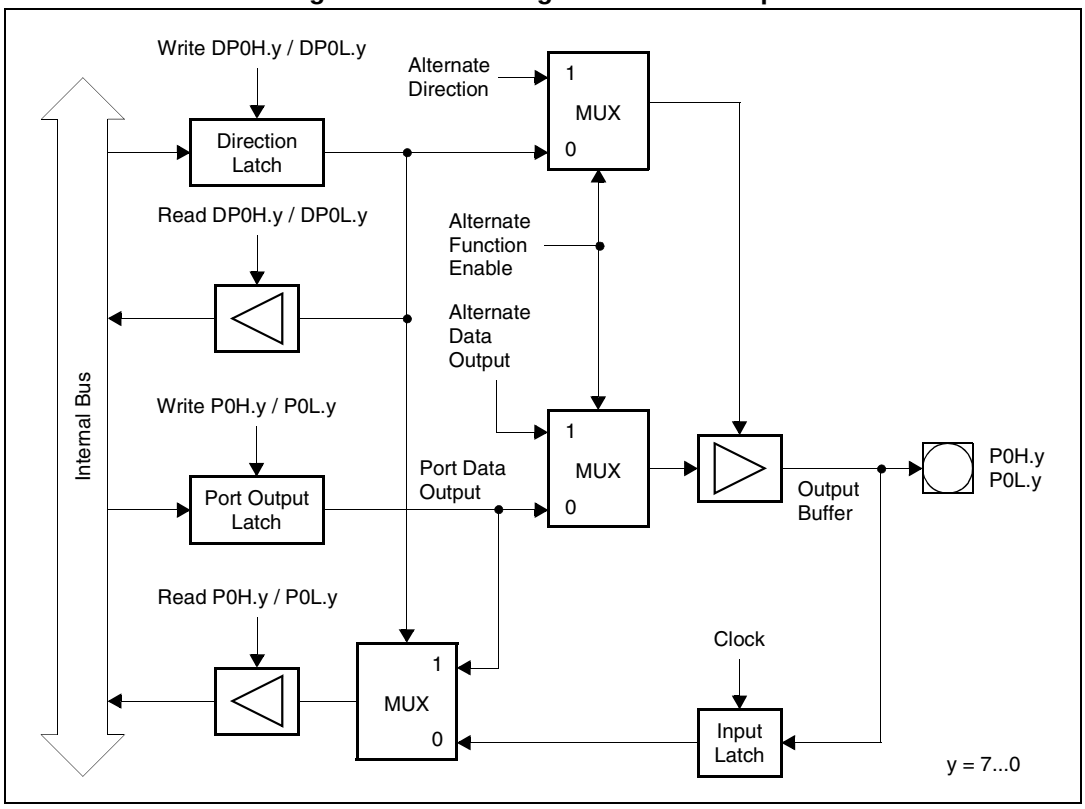
When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled “Alternate Data Output” via 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 28. PORT0 I/O and alternate functions



The *Figure 29* shows the structure of a PORT0 pin.

Figure 29. 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	**Port direction register DP1H or DP1L bit y** '0': Port line P1X.y is an input (high-impedance). '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) are also capture input lines for the CAPCOM2 unit (CC27-24 I).

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

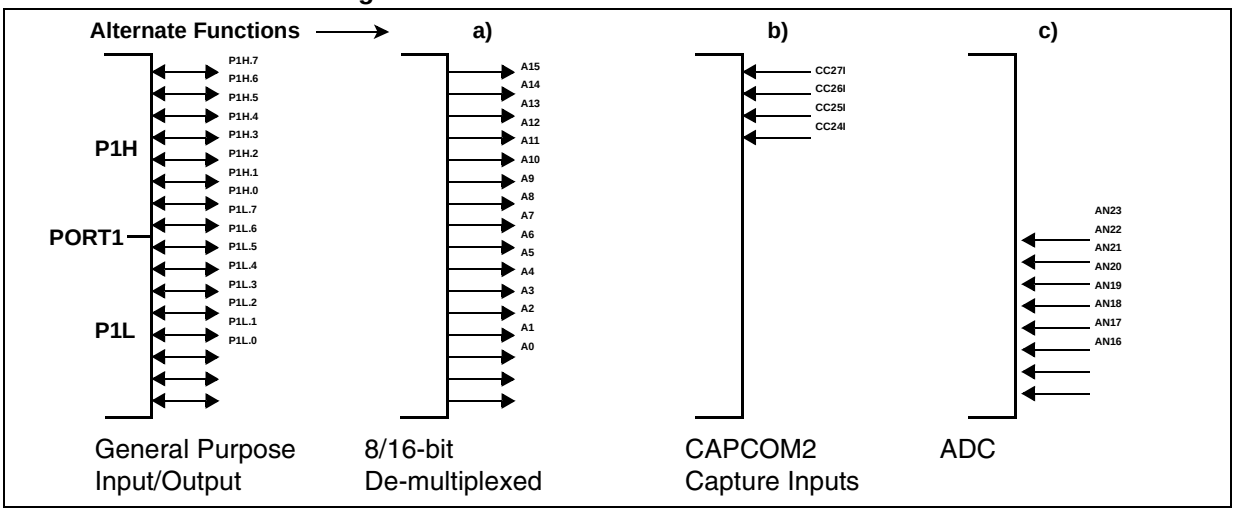
As a side effect, the capture input capability of these lines can also be used in the address bus mode. Hereby changes of the upper address lines could be detected and trigger an interrupt request in order to perform some special service routines. External capture signals can only be applied if no address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

The lower 8 bits of PORT1 (P1L) also serve as input lines for the eight additional analog channels for the A/D converter: General purpose I/O or external memory bus functions must be properly disabled in order to avoid data conflicts when using P1L pins as analog input lines. General purpose I/O and analog input functionality can be mixed without any problem along P1L pins, that is some of P1L pins can be used as general purpose I/O, others as analog input lines; it is also possible (if meaningful at application level) to use the same pin as analog input and general purpose I/O line.

Figure 30. PORT1 I/O and alternate functions



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

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

The *Figure 32* shows the structure of a PORT1 pin.

6.3.2 PORT1 analog inputs disturb protection

A new register is provided for additional disturb protection support on analog inputs for P1L. In particular it allows to disable both the digital input and output sections of the I/O structure. To access this register the bit XMISCEN of register XPERCON and bit XPEN of register SYSCON must be set. Once a bit of the register is set, the corresponding pin can no longer be used as general purpose I/O.

XP1DIDIS (EB36h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	XP1DIDIS.7	XP1DIDIS.6	XP1DIDIS.5	XP1DIDIS.4	XP1DIDIS.3	XP1DIDIS.2	XP1DIDIS.1	XP1DIDIS.0
								RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
XP1DIDIS.y	**PORT1 Digital Disable register bit y** '0': Port line P1.y digital input and output are not disabled: the port pin is defined through the corresponding bits of the standard registers P1L/DP1L. General Purpose Input/Output functionality is available, and also external memory interface functionality. '1': Port line P1.y digital input and output are disabled (necessary for input leakage current reduction and to avoid undesired conflict between output driver configuration and analog input signal). Once this bit is set, P1L/DP1L corresponding bits are no longer effective and the external memory interface functionality is masked on the single bit.

Figure 31. Block diagram of input section of a P1L pin

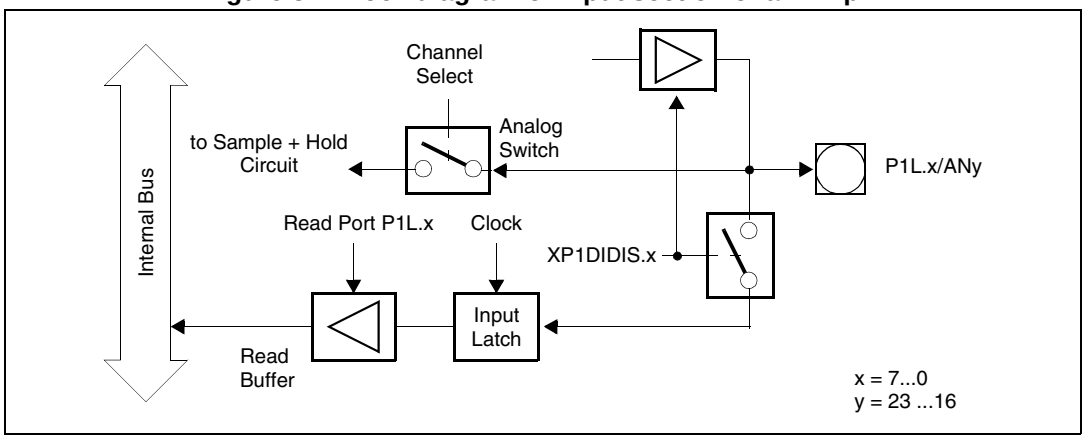
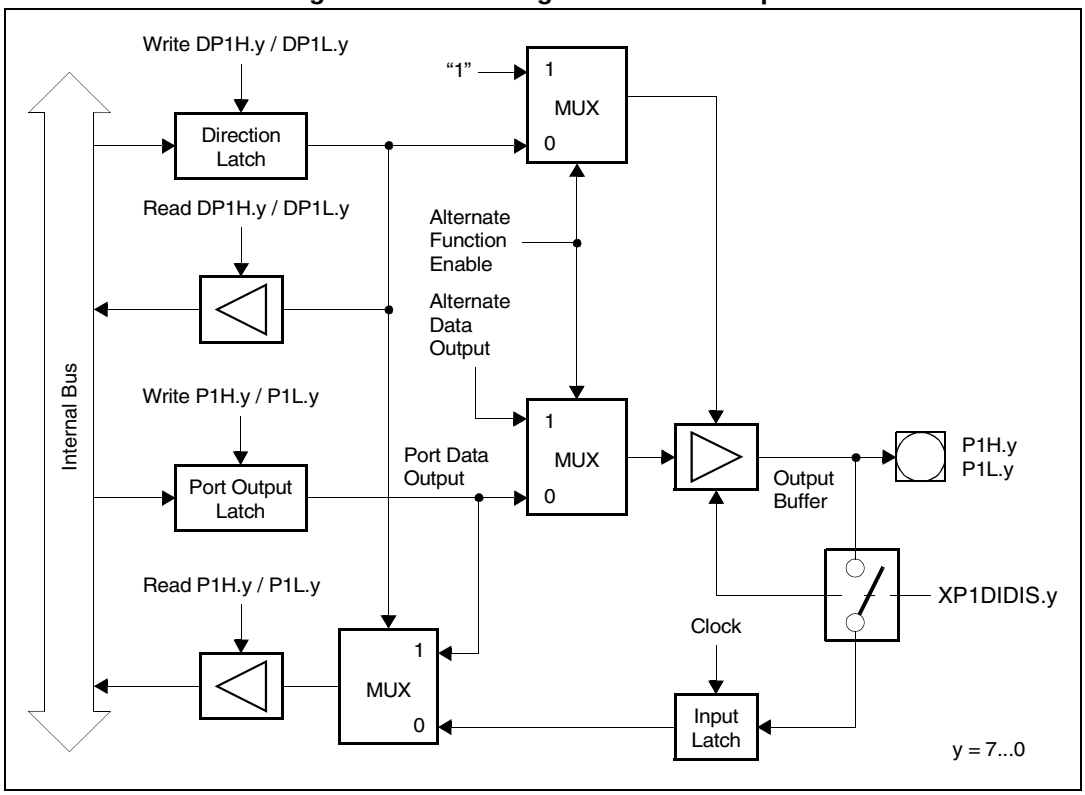


Figure 32. Block diagram of a PORT1 pin



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

Bit	Function
P2.y	Port data register P2 bit y

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

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

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

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

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

Bit	Function
ODP2.y	**Port Open Drain control register ODP2 bit y** '0': Port line P2.y output driver in push-pull mode. '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 2 on page 24*), 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 P2.7...P2.0 can also be used as external interrupt inputs with a sample rate of eight CPU clock cycles.

For pins P2.15 to P2.8, the sampling rate is 8 CPU clock cycles when used as capture input, and 1 CPU clock cycle if used as fast external input.

6.4.2 External interrupts

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

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

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

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

Bit	Function
EXIxSS	**External Interrupt x Source Selection** (x=7...0) '00': Input from associated Port2 pin. '01': Input from "alternate source". '10': Input from Port2 pin ORed with "alternate source". '11': Input from Port2 pin ANDed with "alternate source".

EXIxSS	Port2 pin	Alternate source	
0	P2.8	CAN1_RxD	P4.5
1	P2.9	CAN2_RxD / SCL	P4.4
2	P2.10	RTCSI (Second)	Internal MUX
3	P2.11	RTCAI (Alarm)	Internal MUX
4...7	P2.12...15	Not used (zero)	-

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

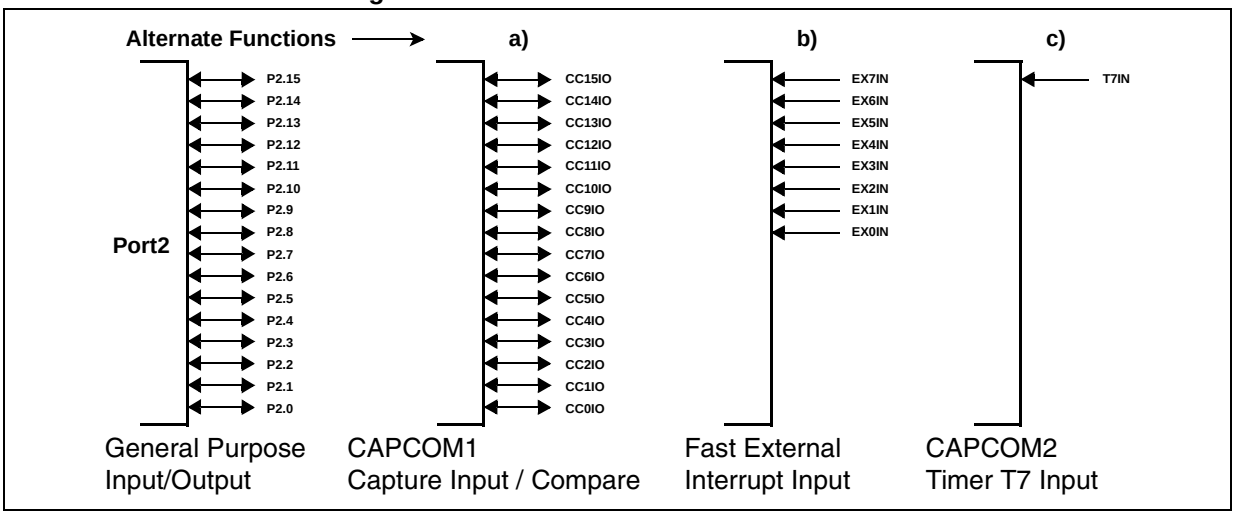
P2.15 in addition is the input for CAPCOM2 timer T7 (T7IN).

The *Table 22* summarizes the alternate functions of Port2.

Table 22. Port2 alternate functions

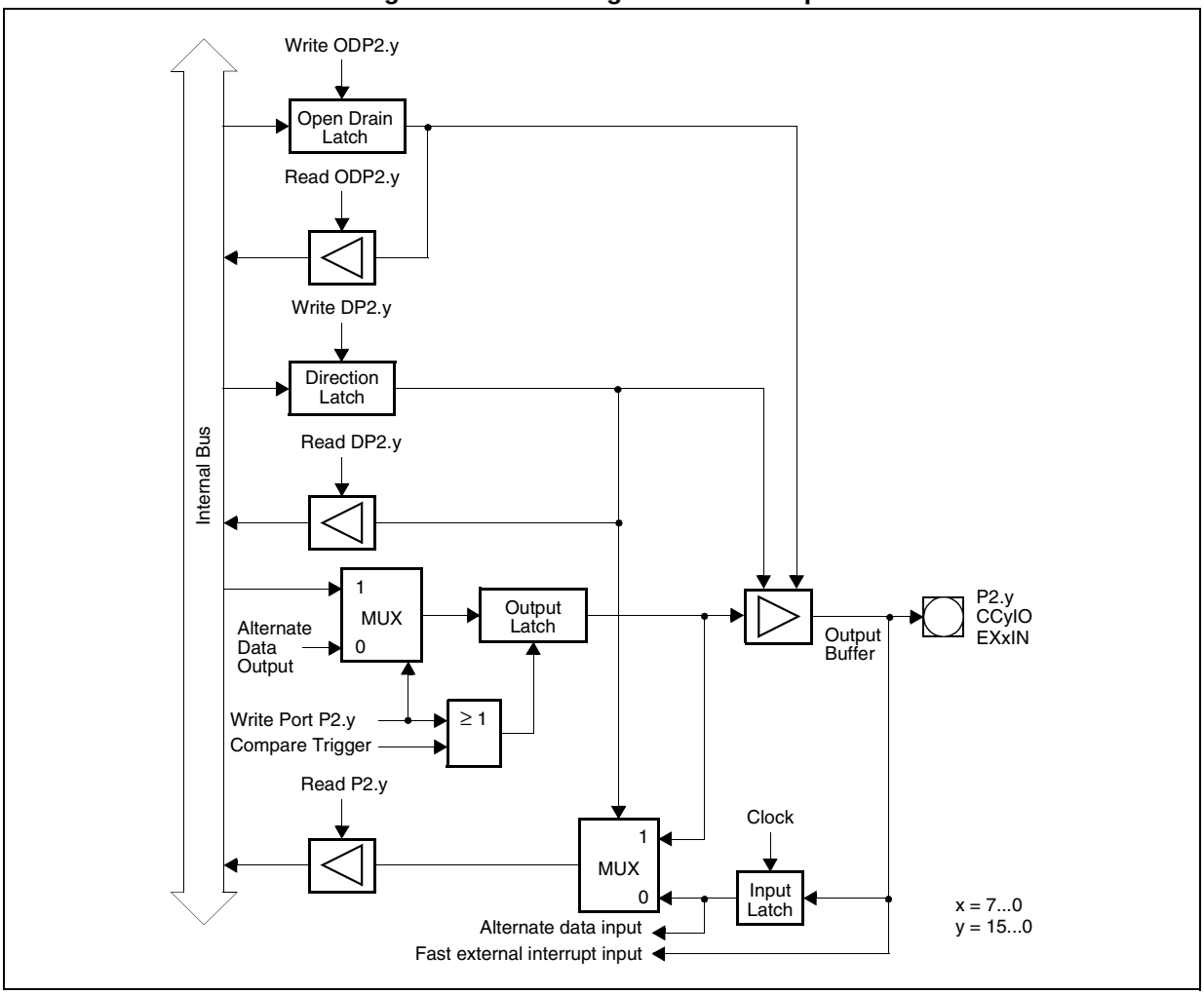
P2 Pin	Alt. Function a)	Alternate function b)		Alternate function c)
P2.0	CC0IO	-		-
P2.1	CC1IO	-		-
P2.2	CC2IO	-		-
P2.3	CC3IO	-		-
P2.4	CC4IO	-		-
P2.5	CC5IO	-		-
P2.6	CC6IO	-		-
P2.7	CC7IO	-		-
P2.8	CC8IO	EX0IN	Fast External Interrupt 0 Input	-
P2.9	CC9IO	EX1IN	Fast External Interrupt 1 Input	-
P2.10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

Figure 33. Port2 I/O and alternate functions



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

Figure 34. Block diagram of a Port2 pin



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 and P3.12 do not support open drain mode).

Due to pin limitations, register bit P3.14 is not connected to any 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** '0': Port line P3.y is an input (high-impedance). '1': Port line P3.y is an output.

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

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

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

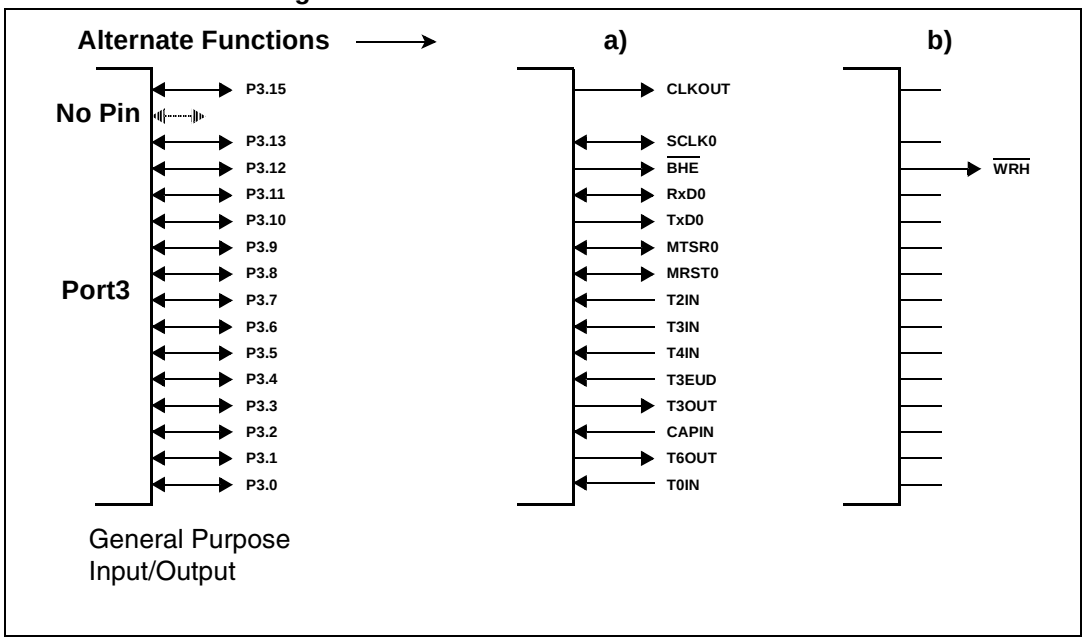
6.5.1 Alternate functions of Port3

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

Table 23. 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	MRST0	SSC Master Receive / Slave Transmit
P3.9	MTSR0	SSC Master Transmit / Slave Receive
P3.10	TxD0	ASC0 Transmit Data Output
P3.11	RxD0	ASC0 Receive Data Input (/ Output in synchronous mode)
P3.12	$\overline{BHE}$/$\overline{WRH}$	Byte High Enable / Write High Output
P3.13	SCLK0	SSC Shift Clock Input/Output
P3.14	---	No pin assigned
P3.15	CLKOUT	System Clock Output (either prescaled or not through register XCLKOUTDIV)

Figure 35. Port3 I/O and alternate functions



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

When the on-chip peripheral associated with a Port3 pin is configured to use the alternate input function, it reads the input latch, which represents the state of the pin, via the line labeled “Alternate Data Input”. Port3 pins with alternate input functions are: T0IN, T2IN, T3IN, T4IN, T3EUD and CAPIN.

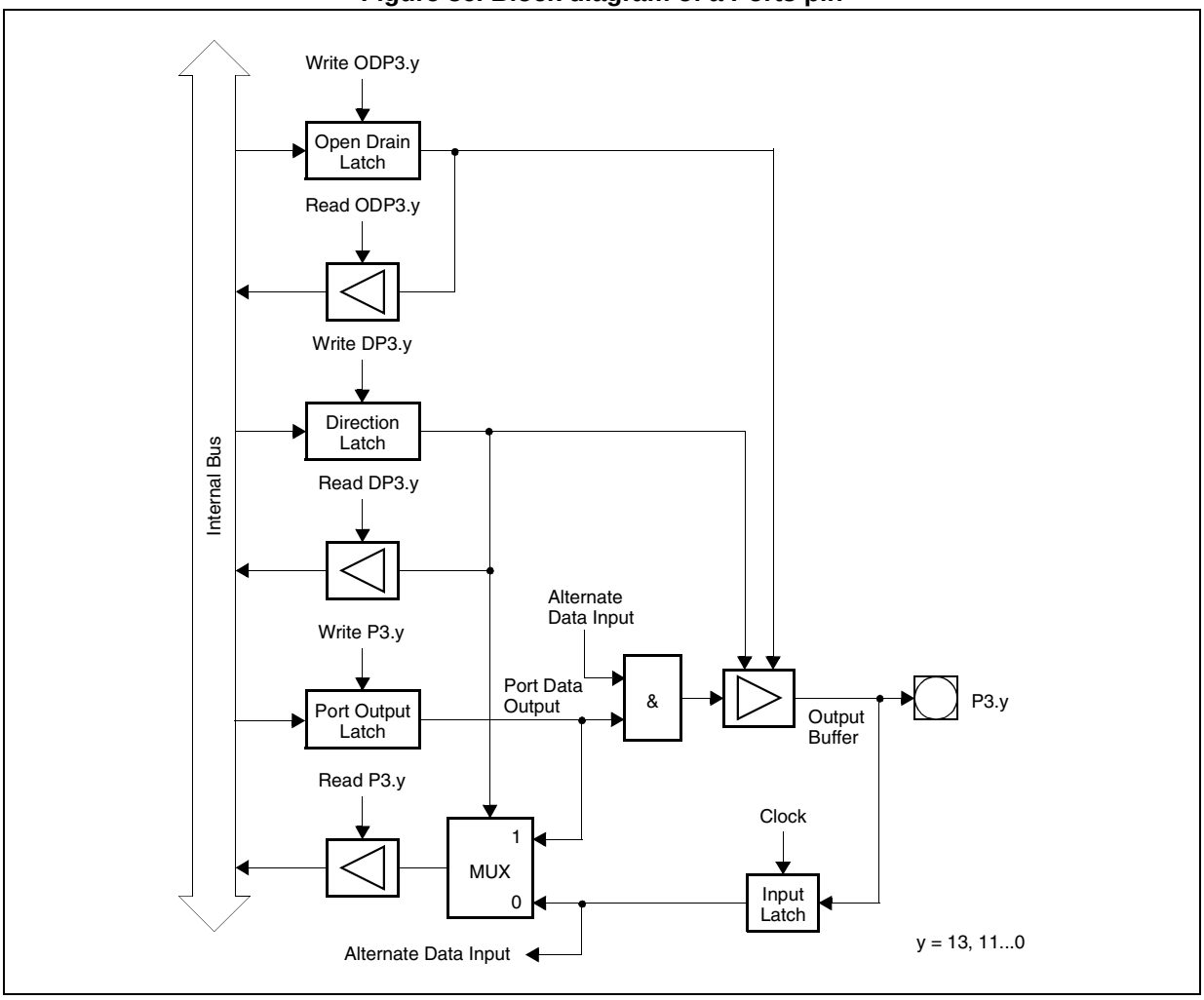
When the on-chip peripheral associated with a Port3 pin is configured to use the alternate output function, its “Alternate Data Output” line is ANDed with the port output latch line.

When using these alternate functions, the user must set the direction of the port line to output (DP3.y = 1) and must set the port output latch (P3.y = 1). Otherwise the pin is in its high-impedance state (when configured as input) or the pin is stuck at '0' (when the port output latch is cleared). When the alternate output functions are not used, the “Alternate Data Output” line is in its inactive state, which is a high level ('1'). Port3 pins with alternate output functions are: T6OUT, T3OUT, TxD0, $\overline{BHE}$ and CLKOUT.

When the on-chip peripheral associated with a Port3 pin is configured to use both the alternate input and output function, the descriptions above apply to the respective current operating mode. The direction must be set accordingly. Port3 pins with alternate input/output functions are: MTSR0, MRST0, RxD0 and SCLK0.

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

Figure 36. Block diagram of a Port3 pin

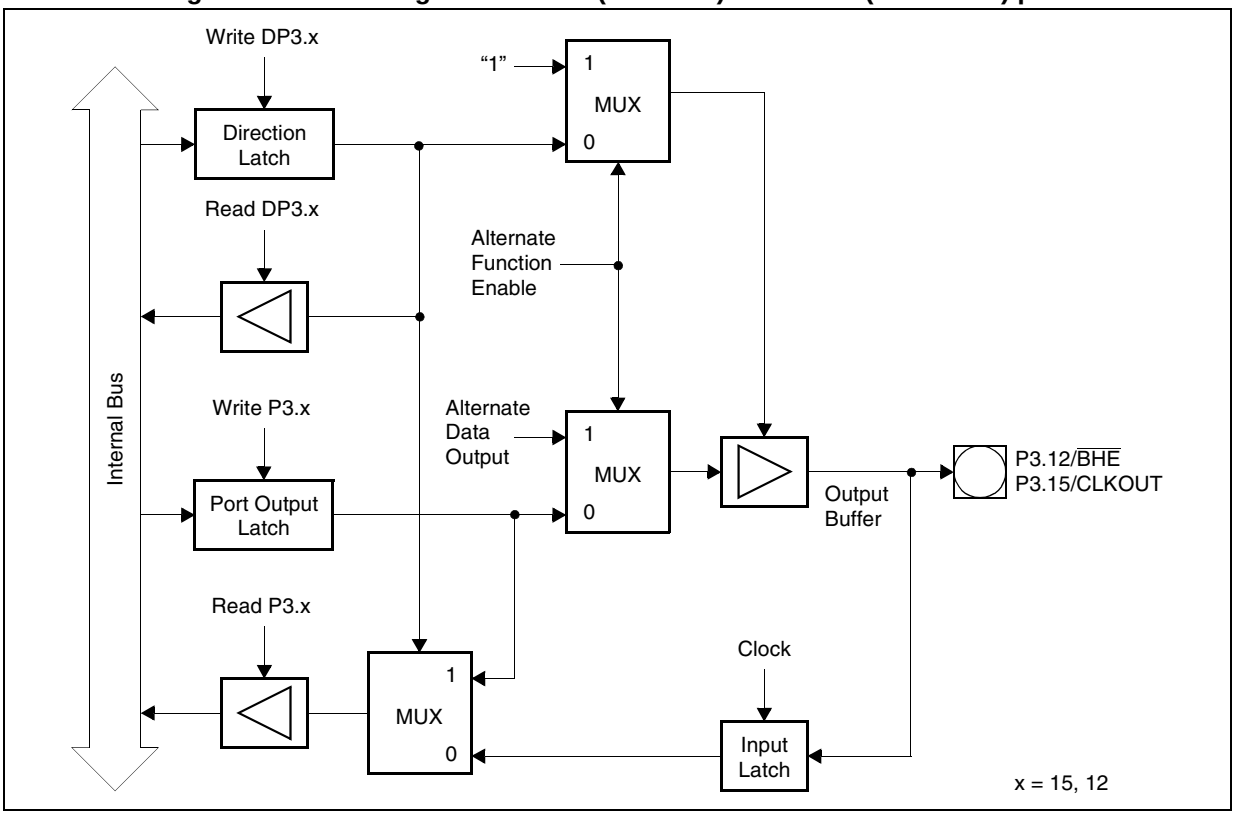


Pin P3.12 ($\overline{BHE}$/$\overline{WRH}$) is another pin with an alternate output function; however, its structure is slightly different (see *Figure 37*). After reset the $\overline{BHE}$ or $\overline{WRH}$ function must be used depending on the system start-up configuration. In either of these cases, there is no possibility to program any port latches before. Thus, the appropriate alternate function is

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

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

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



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

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

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

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

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

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

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

Only bits 4 to 7 are implemented, all other bits will be read as "0".

When I^2C is enabled by setting bit XPEN of the register SYSCON and bit XI2CEN of register XPERCON, pins P4.4 and P4.7 becomes fully dedicated to I^2C interface, and all the other alternate functions are bypassed (external memory and CAN2 functions). The pins are also automatically configured as Open-Drain as requested by the I^2C bus standard. The Port4 control registers P4, DP4 and ODP4 can no longer control P4.7 and P4.4 pins configuration: writing in the bits corresponding to P4.4 and P4.7 in these registers has no effect on pins behavior.

6.6.1 Alternate functions of Port4

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

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

The CAN interfaces use two or four pins of Port4 to interface the CAN Module to the external CAN transceiver. In this case the number of possible segment address lines is reduced. The same should be applied, when the I^2C interface module is used.

The *Table 24* summarizes the alternate functions of Port4 depending on the number of selected segment address lines (coded via bit-field SALSEL).

Table 24. Port4 alternate functions

Port4	Standard function SALSEL = 01 64 Kbytes	Alternate function SALSEL = 11 256 Kbytes	Alternate function SALSEL = 00 1 Mbyte	Alternate function SALSEL = 10 16 Mbytes
P4.0	GPIO	Segment Address A16	Segment. Address A16	Segment Address A16
P4.1	GPIO	Segment Address A17	Segment Address A17	Segment Address A17
P4.2	GPIO	GPIO	Segment Address A18	Segment Address A18
P4.3	GPIO	GPIO	Segment Address A19	Segment Address A19
P4.4	GPIO/CAN2_RxD/SCL	GPIO/CAN2_RxD/SCL	GPIO/CAN2_RxD/SCL	Segment Address A20/SCL
P4.5	GPIO/CAN1_RxD	GPIO/CAN1_RxD	GPIO/CAN1_RxD	Segment Address A21
P4.6	GPIO/CAN1_TxD	GPIO/CAN1_TxD	GPIO/CAN1_TxD	Segment Address A22
P4.7	GPIO/CAN2_TxD/SDA	GPIO/CAN2_TxD/SDA	GPIO/CAN2_TxD/SDA	Segment Address A23/SDA

Relative priority of Port4 alternate functions

When SALSEL = ’10’, CAN1 and CAN2 cannot be used: it means that external memory has higher priority on CAN alternate function. On the contrary, once I^2C is enabled, P4.4 and P4.7 are dedicated to I^2C: it means that I^2C has higher priority on CAN alternate functions and segment address functions as well. If SALSEL = ’10’ (8 segment address lines are enabled) and I^2C is enabled then:

- P4.4 and P4.7 are dedicated to I2C and used as SCL and SDA respectively
- P4.5 and P4.6 continues to output address lines.

Figure 38. Port4 I/O and alternate functions

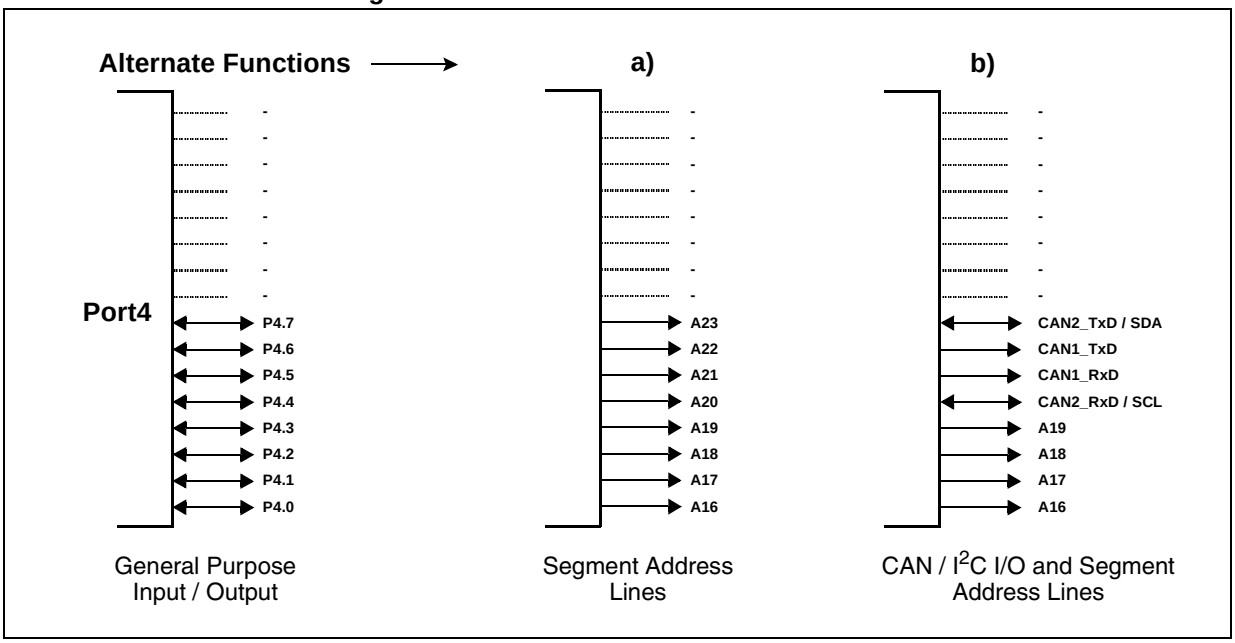


Figure 39. Block diagram of Port4 pins 3...0

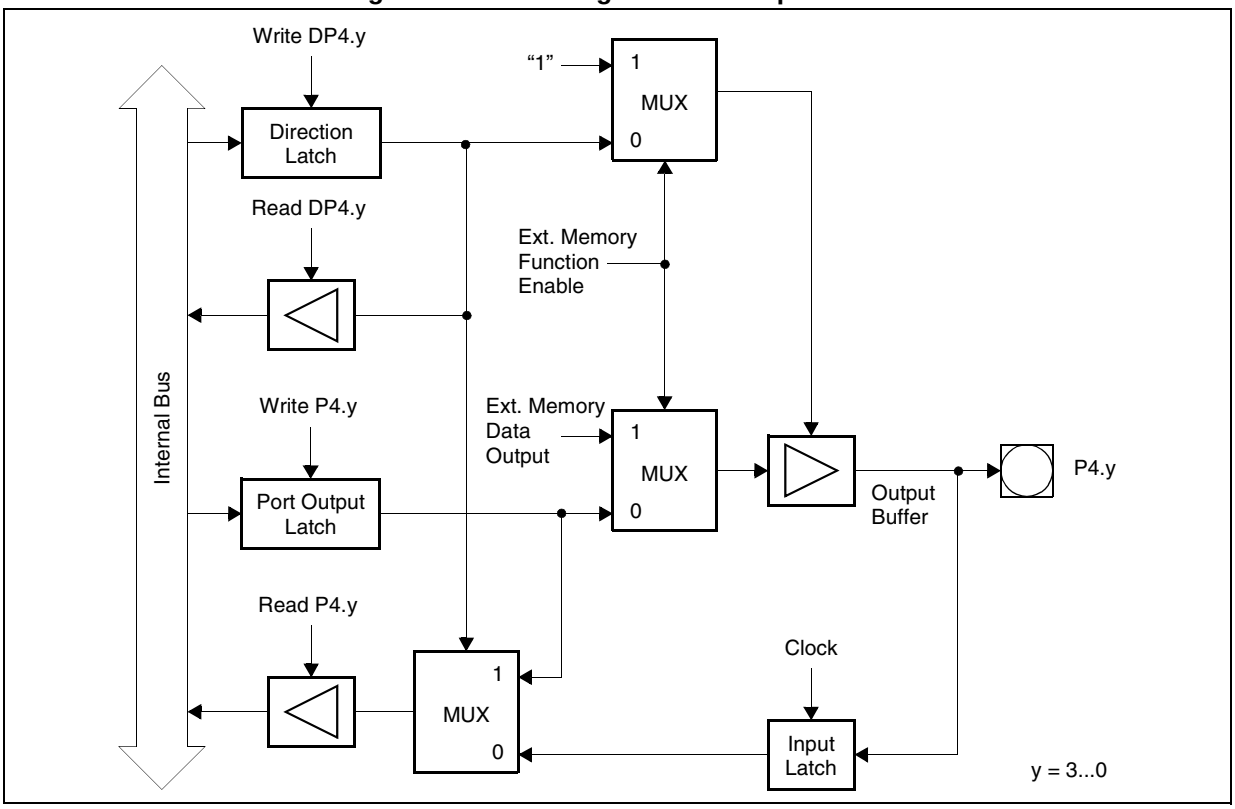
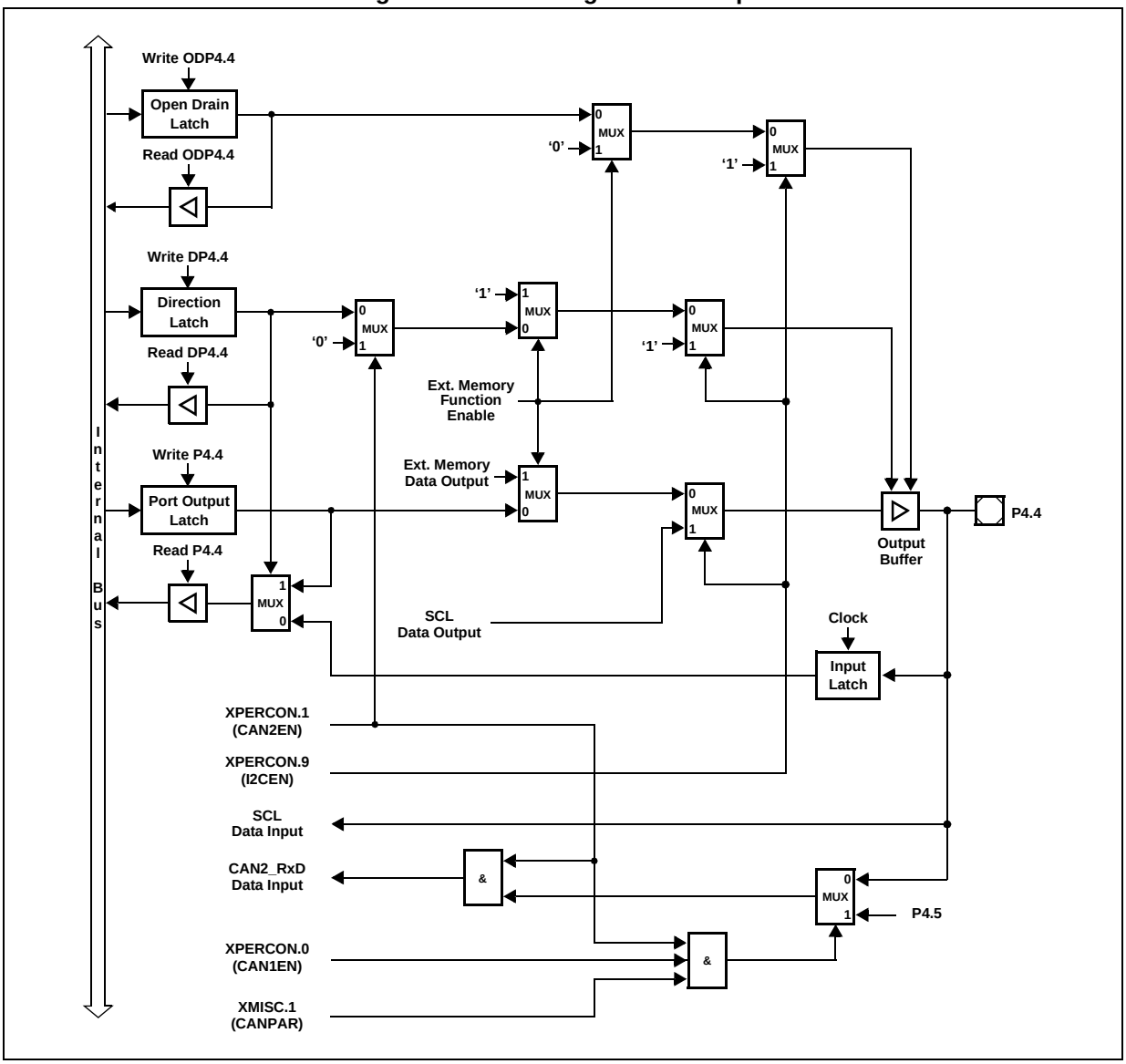
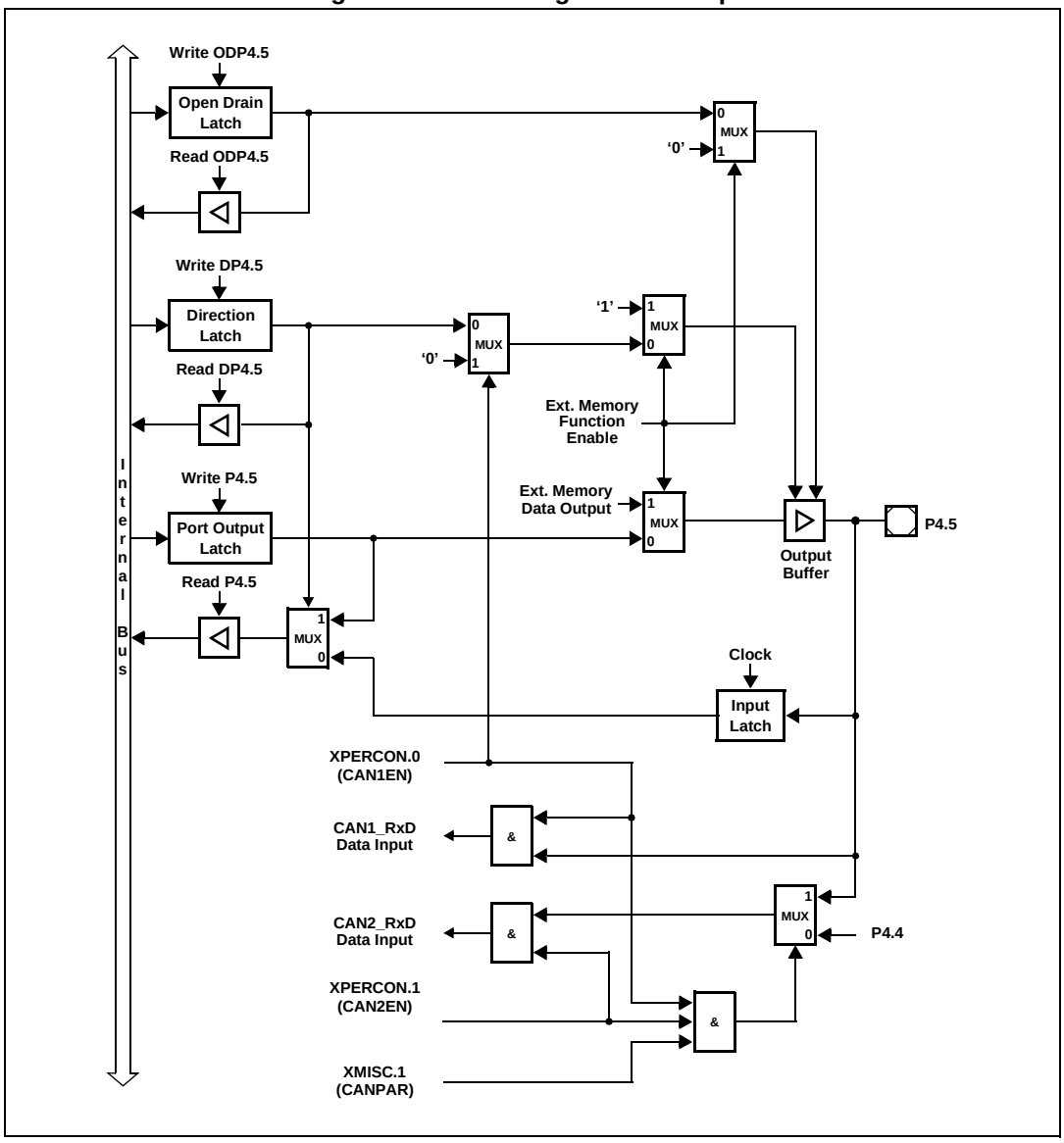


Figure 40. Block diagram of P4.4 pin



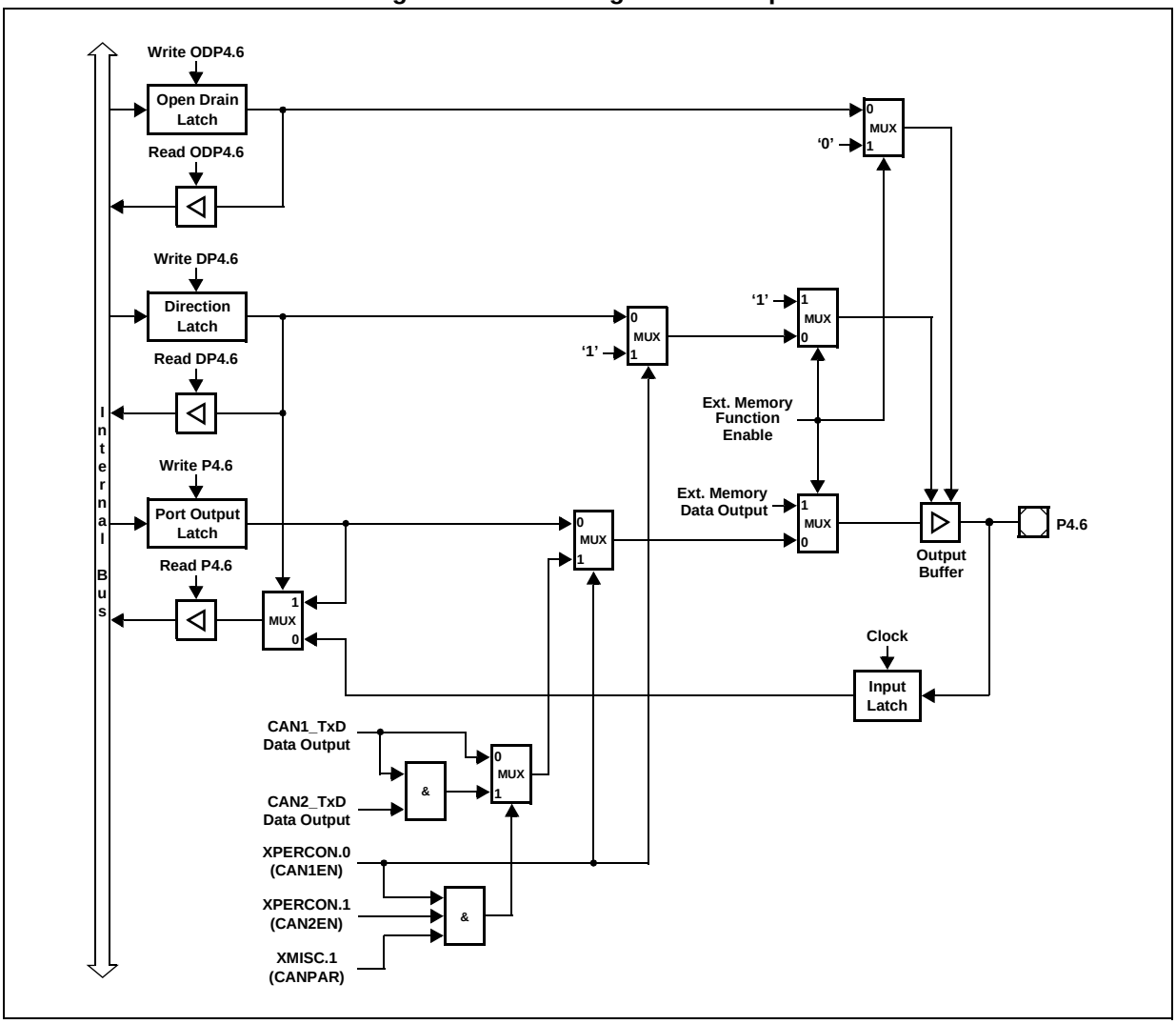
Note: When SALSEL = '10', that is 8-bit segment address lines are selected, P4.4 is dedicated to output the address: any attempt to use the CAN2 on P4.4 is masked. On the contrary, enabling the I^2C, also the segment function is masked; the pin P4.4 is automatically configured as open-drain and used to input and output SCL alternate function.
When CAN parallel mode is selected, CAN2_RxD is remapped on P4.5: this occurs only if CAN1 is enabled as well. On the contrary, if CAN1 is disabled, no remapping occurs.

Figure 41. Block diagram of P4.5 pin



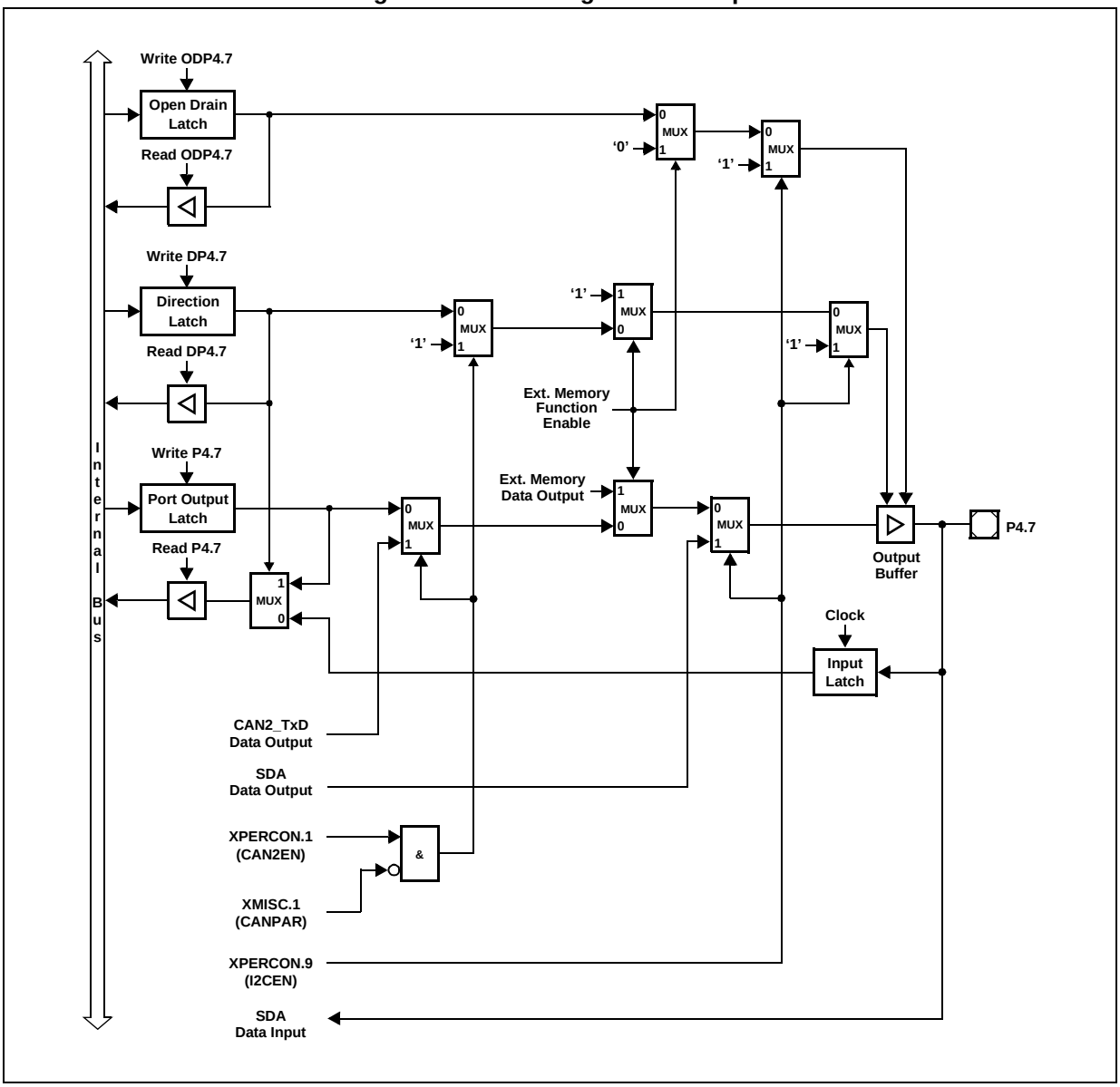
Note: When SALSEL = '10', that is 8-bit segment address lines are selected, P4.5 is dedicated to output the address: any attempt to use the CAN1 on P4.5 is masked.
When CAN parallel mode is selected, CAN2_RxD is remapped on P4.5: this occurs only if CAN1 is enabled as well. On the contrary, if CAN1 is disabled, no remapping occurs.

Figure 42. Block diagram of P4.6 pin



Note: When SALSEL = '10', that is 8-bit segment address lines are selected, P4.6 is dedicated to output the address: any attempt to use the CAN1 on P4.6 is masked.
When CAN parallel mode is selected, CAN2_TxD is remapped on P4.6: this occurs only if CAN1 is enabled as well. On the contrary, if CAN1 is disabled, no remapping occurs.

Figure 43. Block diagram of P4.7 pin



Note: When SALSEL = '10', that is 8-bit segment address lines are selected, P4.7 is dedicated to output the address: any attempt to use the CAN2 on P4.7 is masked. On the contrary, enabling the I^2C, also the segment function is masked; the pin P4.7 is automatically configured as open-drain and used to input and output SDA alternate function.
When CAN parallel mode is selected, CAN2_TxD is remapped on P4.6: this occurs only if CAN1 is enabled as well. On the contrary, if CAN1 is disabled, no remapping occurs.

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

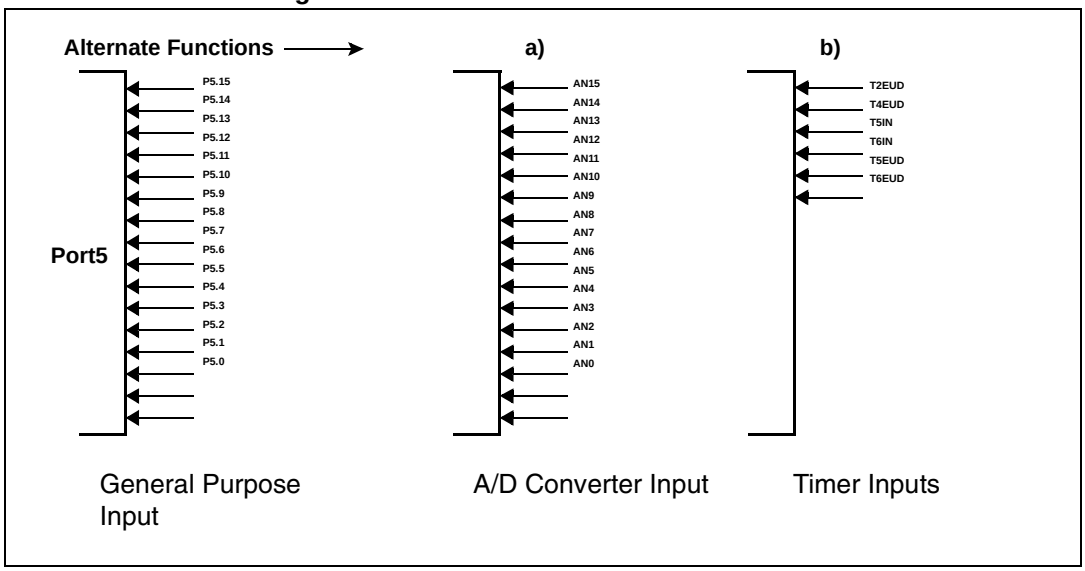
Each line of Port5 is also connected to the input multiplexer of the analog/digital converter. All port lines (P5.15...P5.0) can accept analog signals (AN15...AN0) that can be converted by the ADC. No special programming is required for pins that should be used as analog inputs. The upper six pins of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

The *Table 25* summarizes the alternate functions of Port5.

Table 25. Port5 alternate functions

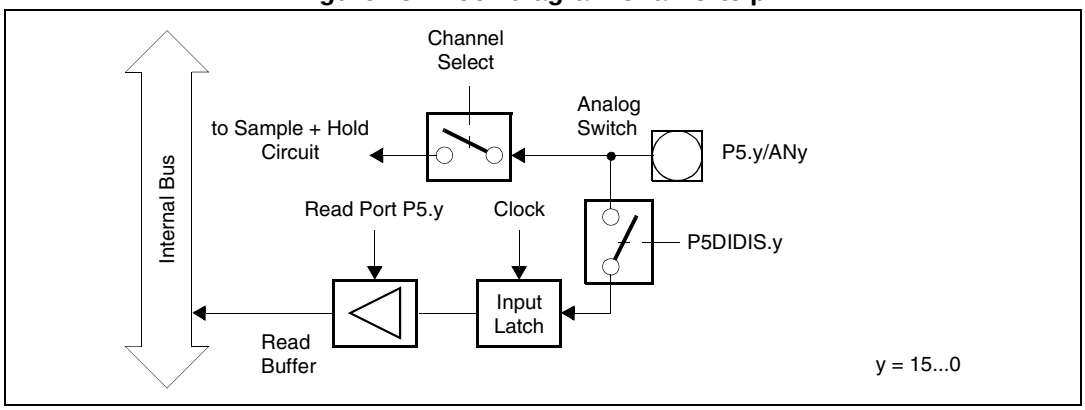
Port5 Pin	Alternate function a)	Alternate function b)
P5.0	Analog Input AN0	-
P5.1	Analog Input AN1	-
P5.2	Analog Input AN2	-
P5.3	Analog Input AN3	-
P5.4	Analog Input AN4	-
P5.5	Analog Input AN5	-
P5.6	Analog Input AN6	-
P5.7	Analog Input AN7	-
P5.8	Analog Input AN8	-
P5.9	Analog Input AN9	-
P5.10	Analog 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 44. Port5 I/O and alternate functions



Port5 pins have a special port structure (see *Figure 45*), 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 45. Block diagram of a Port5 pin



6.7.2 Port5 analog inputs disturb protection

A Schmitt trigger protection can be activated on each pin of Port5 by setting the dedicated bit of register P5DIDIS.

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

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

Bit	Function
P5DIDIS.y	**Port5 Digital Disable register bit y** '0': Port line P5.y digital input is enabled (Schmitt trigger enabled). '1': Port line P5.y digital input is disabled (Schmitt trigger disabled, necessary for input leakage current reduction).

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.

Since in ST10F276 the XSSC is implemented on P6.5...P6.7, when the module is enabled through the XPERCON register, the corresponding bits of P6, DP6 and ODP6 are overwritten by the new XSSCPORT register (mapped on XBUS), which again allows the user to program pin P6.5...P6.7 according to the XSSC configuration.

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

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

Bit	Function
P6.y	Port data register P6 bit y

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

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

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

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

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

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

XSSCPORT (E880h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-							XOD P6.7	XP6 .7	XDP6 .7	XOD P6.6	XP6 .6	XDP6 .6	XOD P6.5	XP6 .5	XDP6 .5
							RW	RW	RW	RW	RW	RW	RW	RW	RW

This register is enabled and visible only when bit XPEN of SYSCON is set, and bit XSSCEN in XPERCON is set as well. On the contrary, the standard P6, DP6 and ODP6 registers must be used to configure pins P6.5, P6.6 and P6.7 when XSSC is disabled.

Bit	Function
XDP6.y	**Port direction register XDP6 bit y (y = 5, 6, 7 only)** '0': Port line P6.y is an input (high-impedance). '1': Port line P6.y is an output.
XP6.y	Port data register bit XP6 y (y = 5, 6, 7 only)
XODP6.y	**Port Open Drain control register XODP6 bit y (y = 5, 6, 7 only)** '0': Port line P6.y output driver in push/pull mode. '1': Port line P6.y output driver in open drain mode.

6.8.1 Alternate functions of Port6

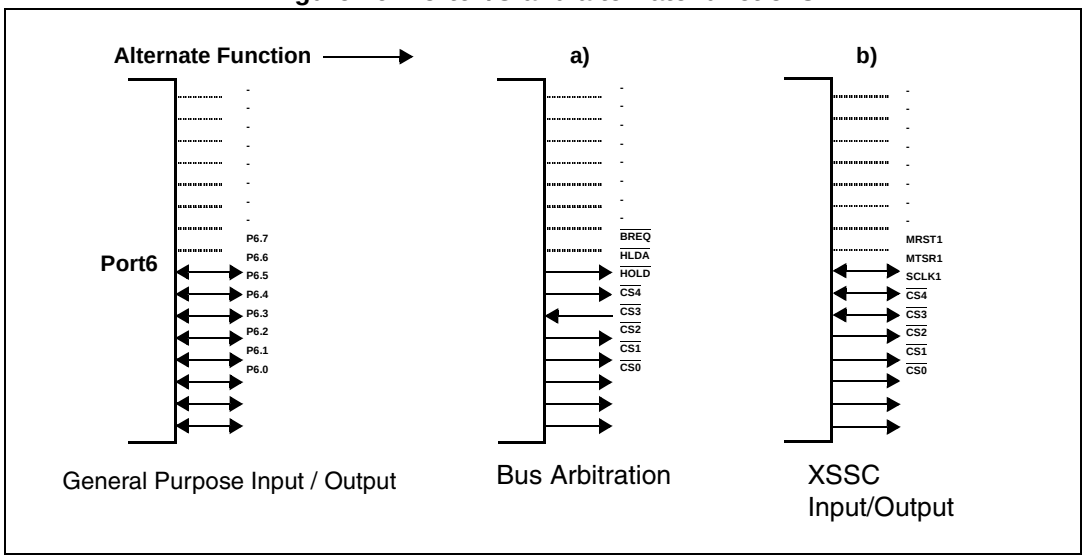
A programmable number of chip select signals (CS4...CS0) derived from the bus control registers (BUSCON4...BUSCON0) can be output on five pins of Port6. The other three pins may be used for bus arbitration to accommodate additional masters in a ST10F276 system. The number of chip select signals is selected via PORT0 during reset. The selected value can be read from bit-field CSSEL in register RP0H (read only) in order to check the configuration during run time.

The *Table 26* summarizes the alternate functions of Port6 depending on the number of selected chip select lines (coded via bit-field CSSEL).

Table 26. 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	HOLD External hold request input / SCLK1			
P6.6	HLDA Hold acknowledge output / MTSR1			
P6.7	$\overline{BREQ}$ Bus request output / MRST1			

Figure 46. 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 ST10F276 is in Hold mode (invoked through $\overline{HOLD}$ pin), 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 ST10F276 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 $\overline{CS}$ 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 $\overline{CS}$ function must be used, if selected so. In this case there is no possibility to program any port latches before. Thus the alternate function ($\overline{CS}$) is selected automatically in this case.

Note: The open drain output option can only be selected via software earliest during the initialization routine; at least signal CS0 will be in push-pull output driver mode directly after reset (see Figure 47).

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 3 on page 30* and *Figure 4 on page 41*).

Figure 47. Block diagram of Port6 pins 4...0

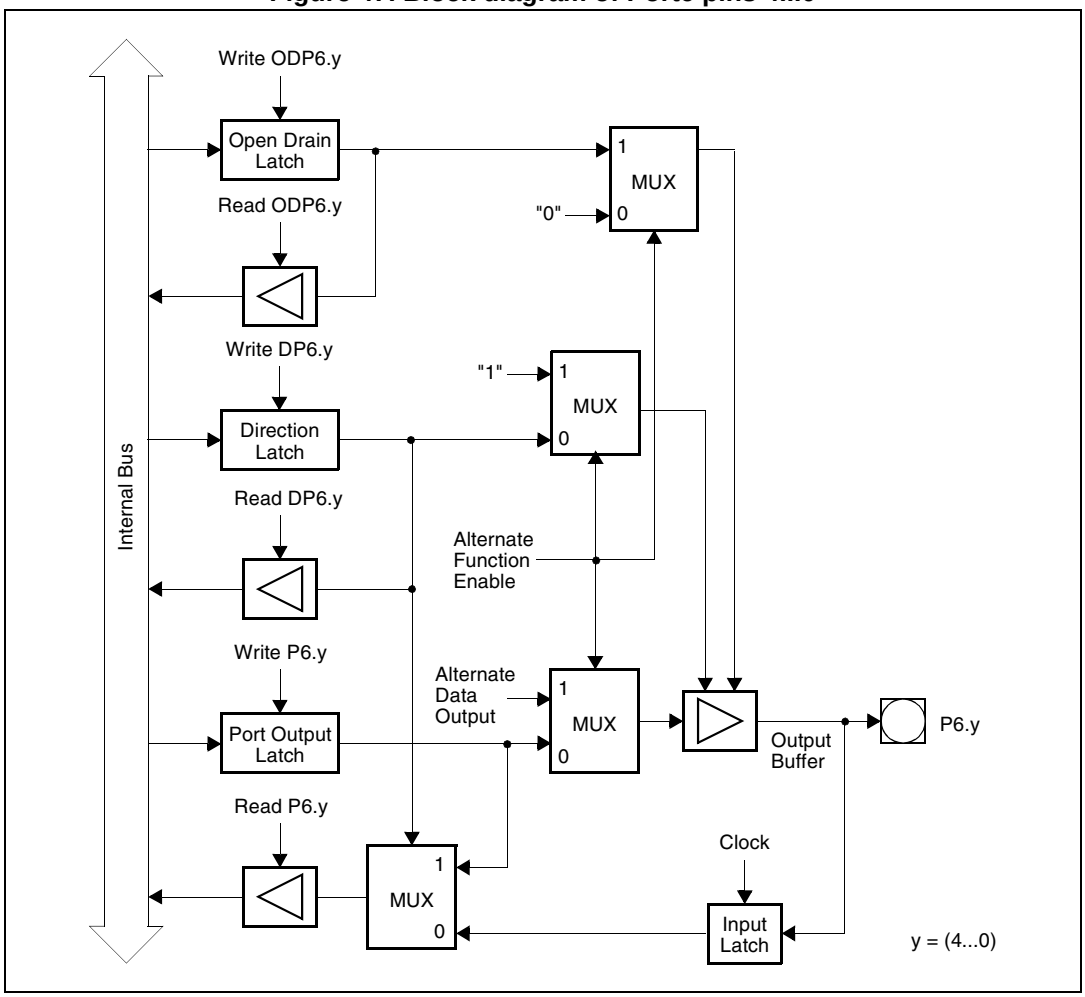
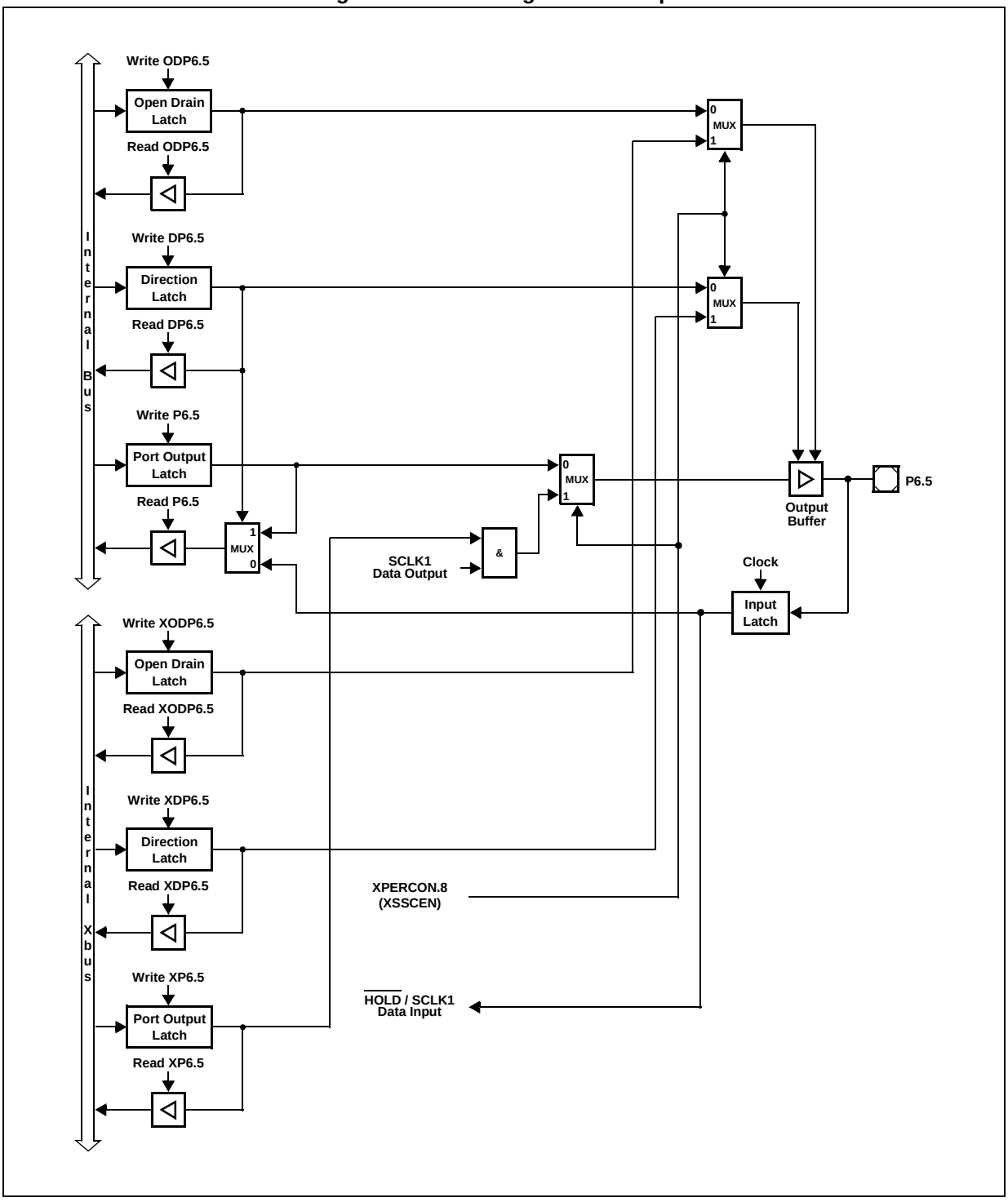


Figure 48. Block diagram of P6.5 pin



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
-	-	-	-	-	-	-	-	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0
								RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
P7.y	Port data register P7 bit y

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

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

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

ODP7 (F1D2h / E9h) ESFR Reset Value: - - 00h

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

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

6.9.1 Alternate functions of Port7

The upper four 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 eight CPU clock cycles.

The lower four 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').

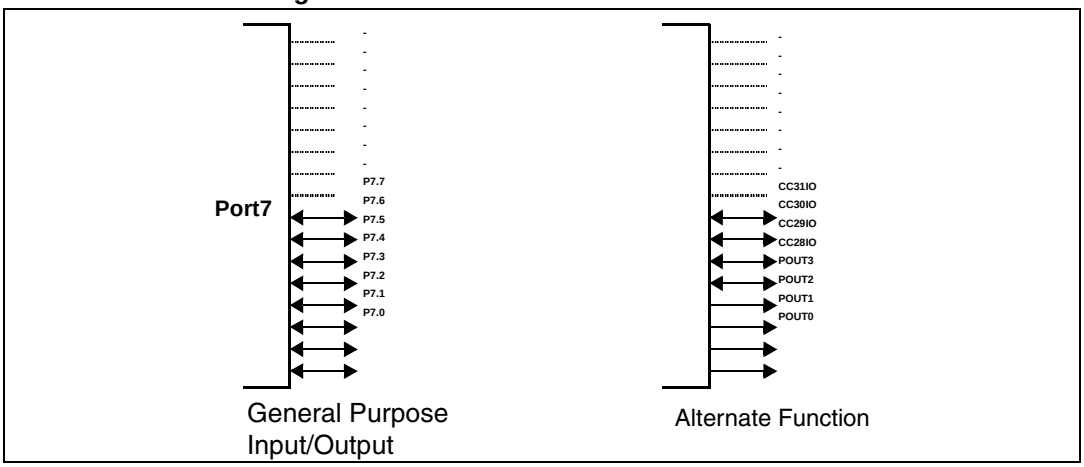
Note that the PWM outputs must be enabled via the respective PENx bit in PWMCON1.

The *Table 27* summarizes the alternate functions of Port7.

Table 27. 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

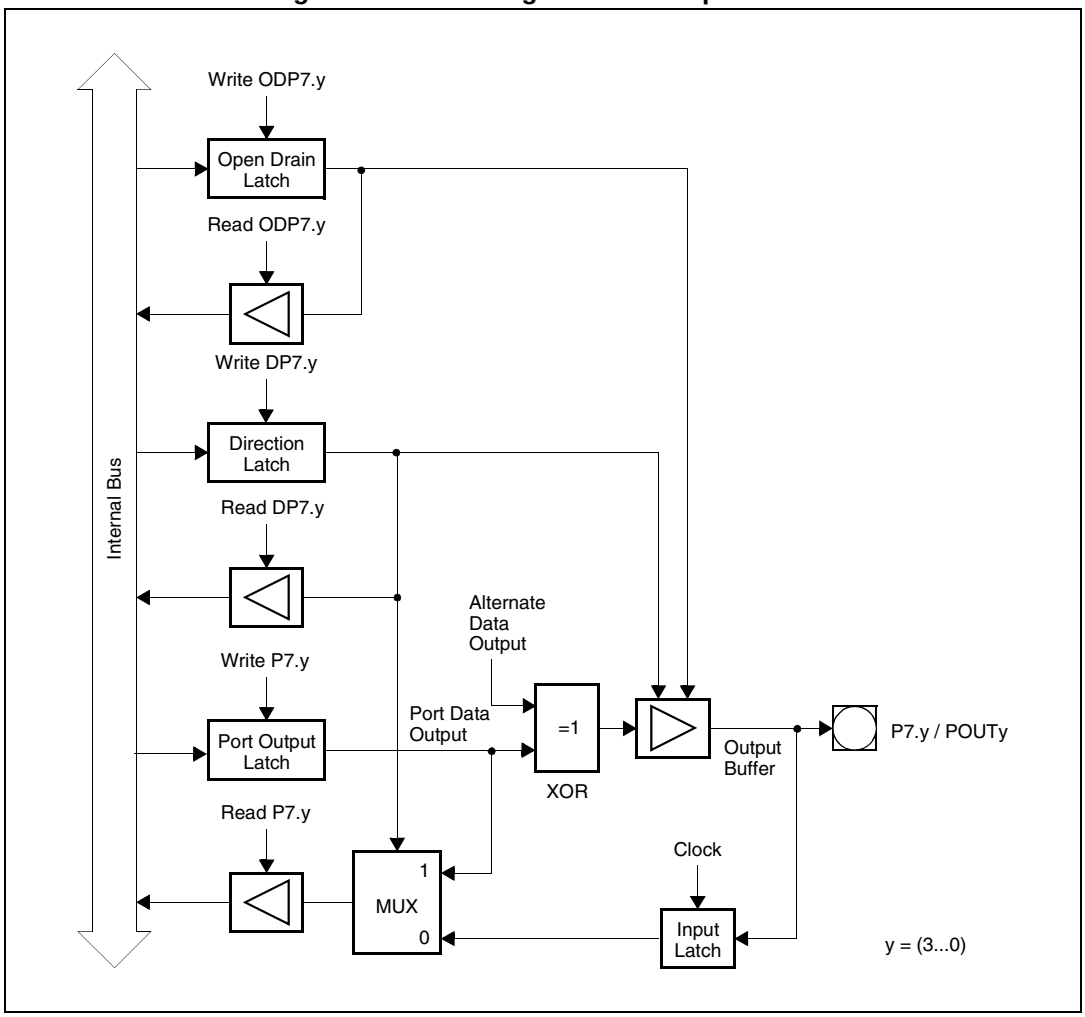
Figure 49. Port7 I/O and alternate functions



The structure of Port7 differs in the way the output latches are connected to the internal bus and to the pin driver (see *Figure 2 on page 24* and *Figure 3 on page 30*).

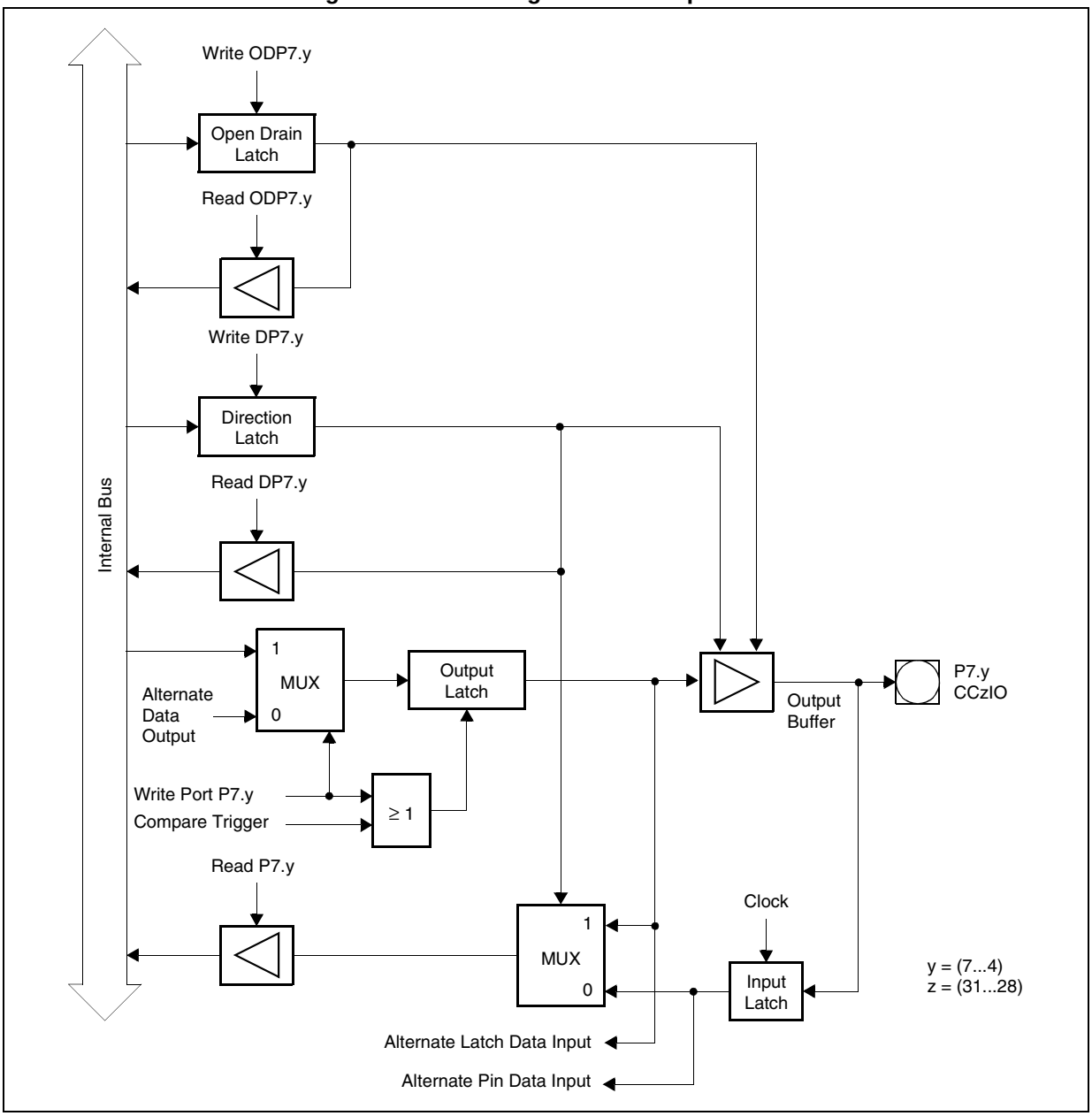
Pins P7.3...P7.0 (POUT3...POUT0) XOR the alternate data output with the port latch output, which allows to use the alternate data directly or inverted at the pin driver.

Figure 50. Block diagram of Port7 pins 3...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 51. Block diagram of Port7 pins 7...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.

Since in ST10F276 the XPWM (or PWM1) and XASC (or ASC1) are implemented on P8.0-P8.3 and P8.6-P8.7 respectively, when these modules are enabled through the XPERCON register, the corresponding bits of P8, DP8 and ODP8 are overwritten by the new

XPWMPORT and XS1PORT registers (mapped on XBUS) which again allows the user to program pin P8.0-P8.3 and P8.6-P8.7 respectively, according to the XPWM an XASC configurations.

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

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

Bit	Function
P8.y	Port data register P8 bit y

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

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

Bit	Function
DP8.y	**Port direction register DP8 bit y** '0': Port line P8.y is an input (high-impedance). '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	Port Open Drain control register ODP8 bit y '0': Port line P8.y output driver in push-pull mode. '1': Port line P8.y output driver in open drain mode.

XPWMPORT (EC80h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-				XODP8.3	XP8.3	XDP8.3	XODP8.2	XP8.2	XDP8.2	XODP8.1	XP8.1	XDP8.1	XODP8.0	XP8.0	XDP8.0
-				RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

This register is enabled and visible only when bit XPEN of SYSCON is set, and bit XPWMEN in XPERCON is set as well. On the contrary, the standard P8, DP8 and ODP8 registers must be used to configure pins P8.0, P8.1, P8.2 and P8.3 when XPWM is disabled.

Bit	Function
XDP8.y	Port direction register bit y (y = 0, 1, 2, 3 only) ‘0’: Port line P8.y is an input (high-impedance). ‘1’: Port line P8.y is an output.
XP8.y	Port data register bit y (y = 0, 1, 2, 3 only)
XODP8.y	Port Open Drain control register bit y (y = 0, 1, 2, 3 only) ‘0’: Port line P8.y output driver in push/pull mode. ‘1’: Port line P8.y output driver in open drain mode.

XS1PORT (E980h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-										XODP8.7	XP8.7	XDP8.7	XODP8.6	XP8.6	XDP8.6
-										RW	RW	RW	RW	RW	RW

This register is enabled and visible only when bit XPEN of SYSCON is set, and bit XASCEN in XPERCON is set as well. On the contrary, the standard P8, DP8 and ODP8 registers must be used to configure pins P8.6 and P8.7 when XASC is disabled.

Bit	Function
XDP8.y	**Port direction register bit y** (y = 6, 7 only) ‘0’: Port line P8.y is an input (high-impedance) ‘1’: Port line P8.y is an output
XP8.y	**Port data register bit y** (y = 6, 7 only)
XODP8.y	**Port Open Drain control register bit y** (y = 6, 7 only) ‘0’: Port line P8.y output driver in push/pull mode ‘1’: Port line P8.y output driver in open drain mode

6.10.1 Alternate functions of Port8

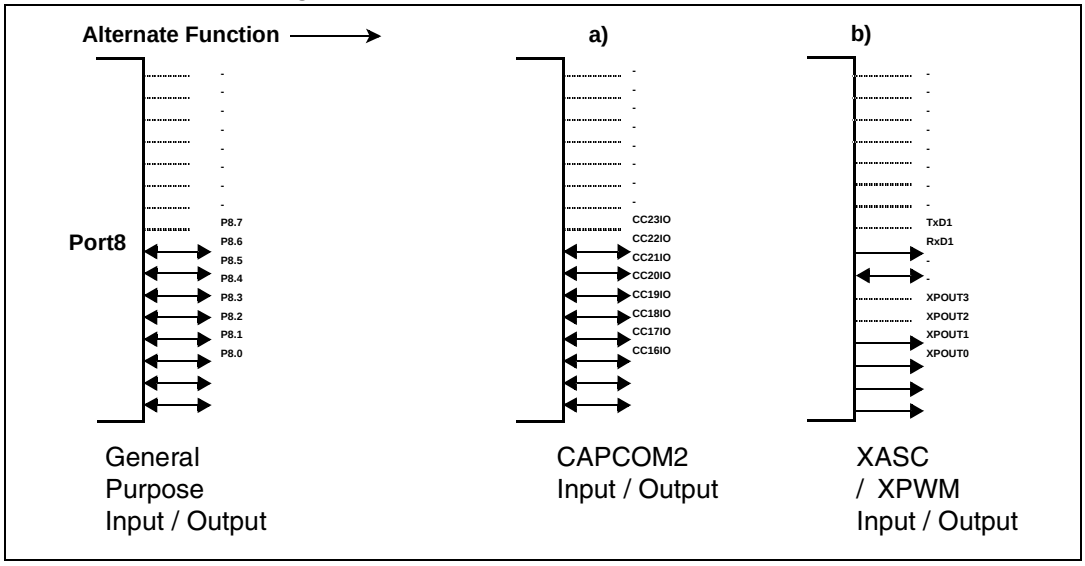
All Port8 lines (P8.7...P8.0) support capture inputs or compare outputs (CC23IO...CC16IO) for the CAPCOM2 unit (see *Table 28*). The use of the port lines by the CAPCOM unit, its accessibility via software and the precautions are the same as described for the Port2 lines.

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 eight CPU clock cycles.

Table 28. Port8 alternate functions

Port8 pin	Alternate function a)	Alternate function b)
P8.0	CC16IO Capture input / compare output ch. 16	XPOUT0XPWM channel 0 output
P8.1	CC17IO Capture input / compare output ch. 17	XPOUT1XPWM channel 1 output
P8.2	CC18IO Capture input / compare output ch. 18	XPOUT2XPWM channel 2 output
P8.3	CC19IO Capture input / compare output ch. 19	XPOUT3XPWM channel 3 output
P8.4	CC20IO Capture input / compare output ch. 20	-
P8.5	CC21IO Capture input / compare output ch. 21	-
P8.6	CC22IO Capture input / compare output ch. 22	RxD1XASC Receive Data Input/Output
P8.7	CC23IO Capture input / compare output ch. 23	TxD1XASC Transmit Data Output

Figure 52. 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 53. Block diagram of Port8 pins 3...0

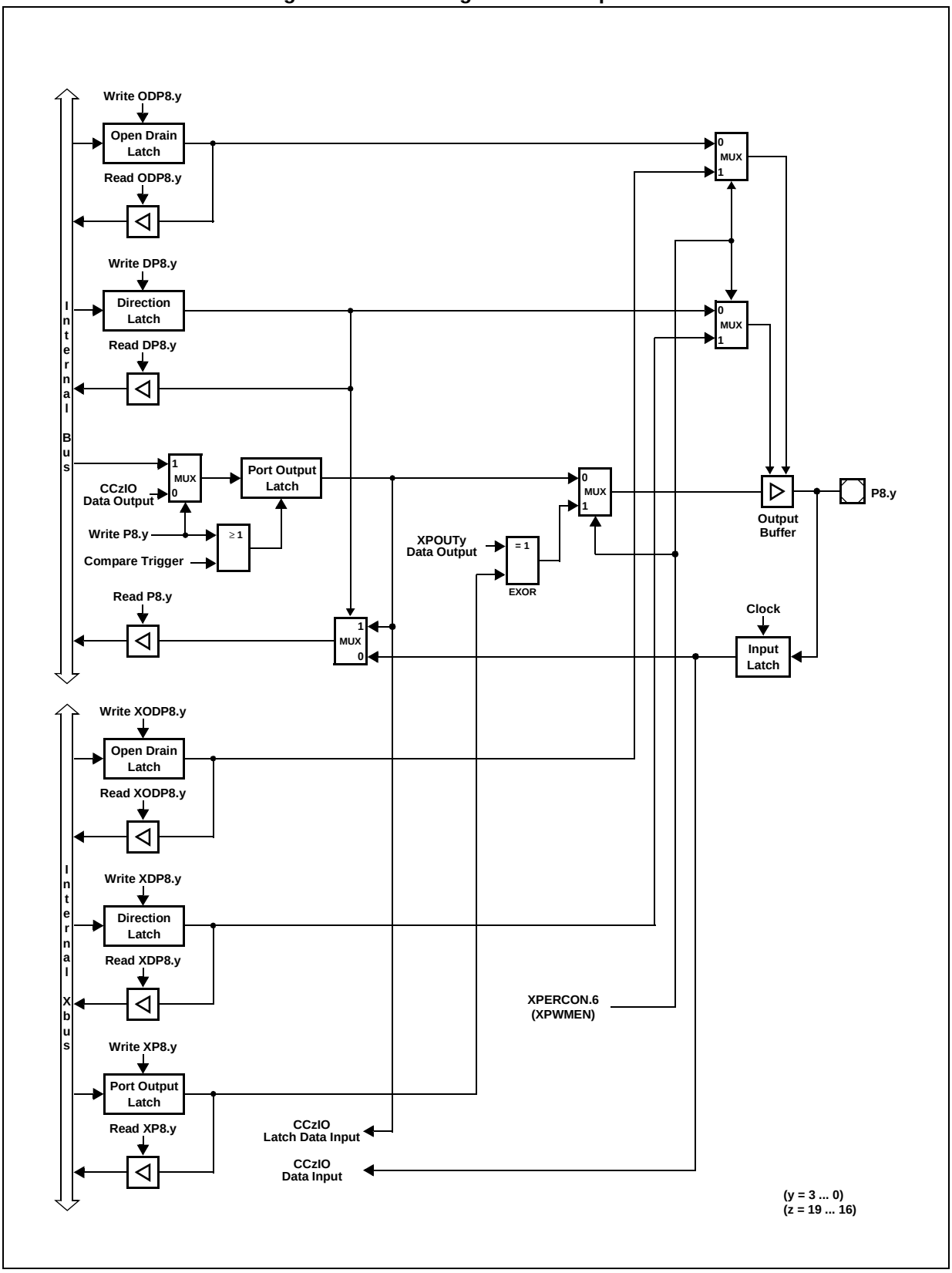


Figure 54. Block diagram of P8.4 and P8.5 pins

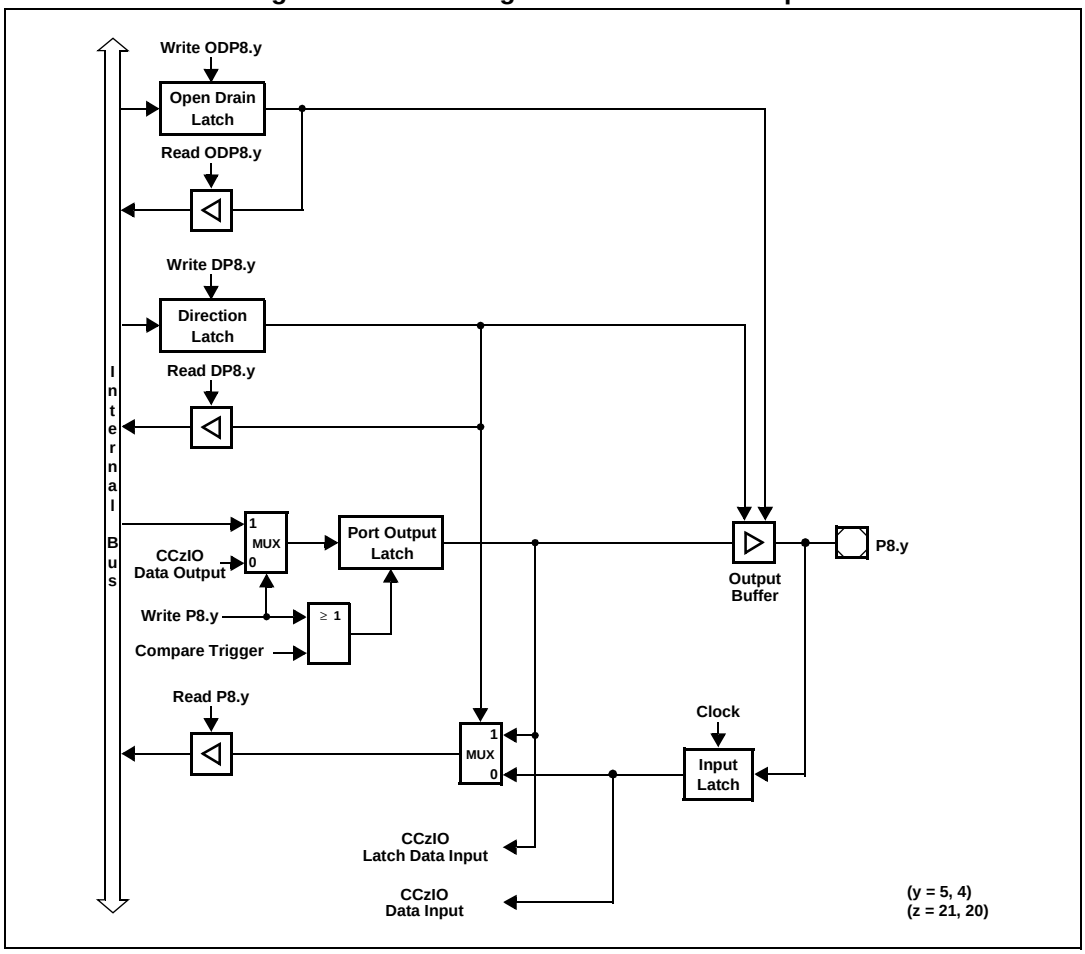
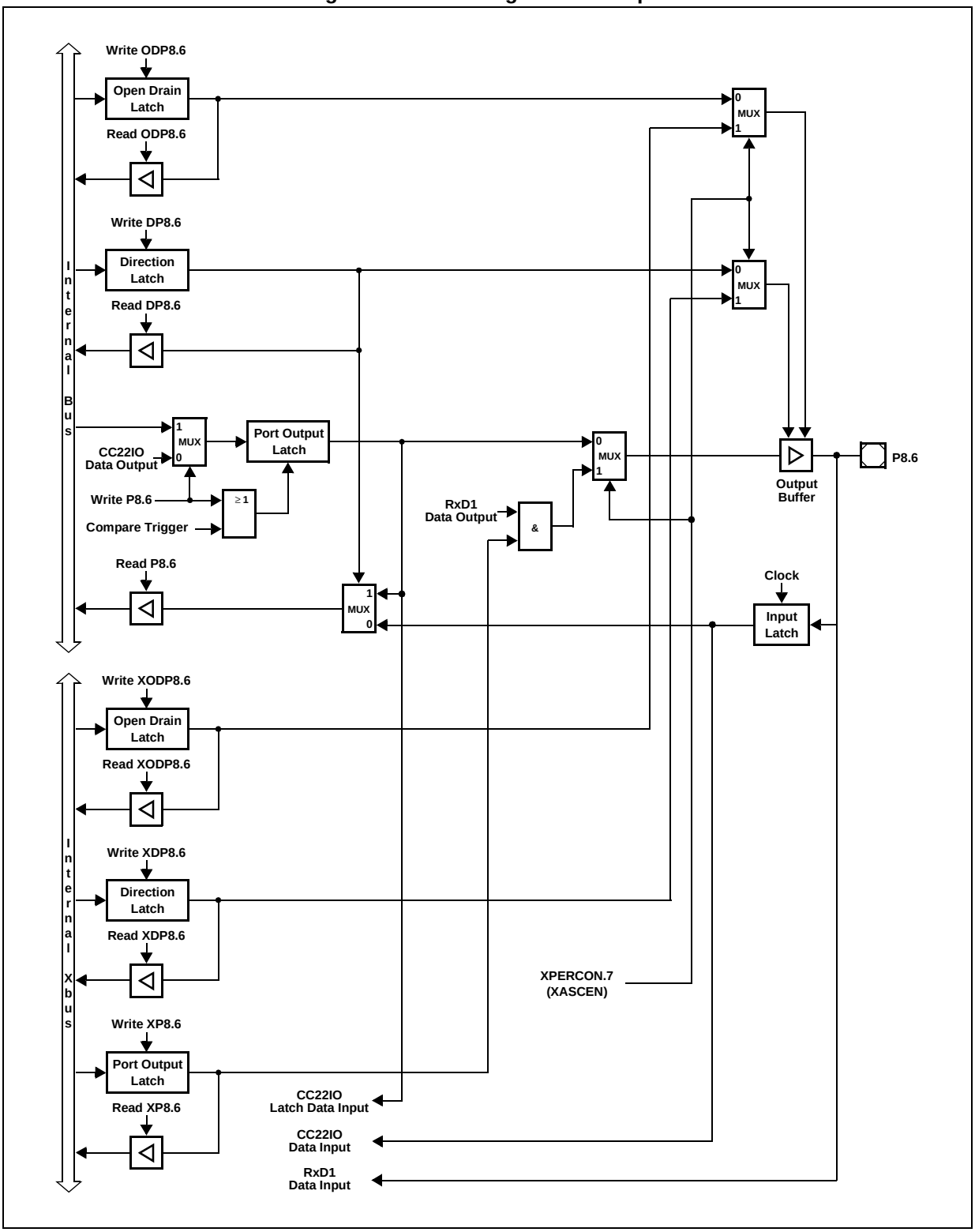


Figure 55. Block diagram of P8.6 pin



7 Dedicated pins

Most of the input/output or control signals of the ST10F276 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 29* summarizes the dedicated pins of the ST10F276.

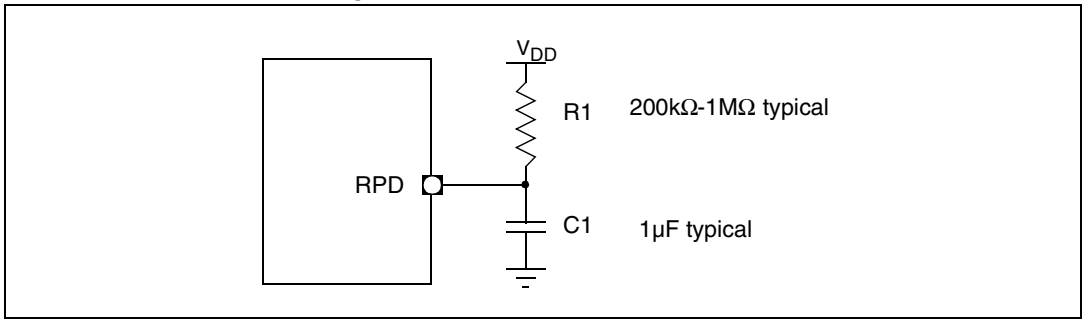
Table 29. Summary of dedicated pins

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 IFlash, to the IRAM 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. During reset, during Hold mode and during Adapt mode an internal pull-down ensures an inactive (low) level on the ALE output.
RD	**External Read Strobe:** controls the output drivers of external memory or peripherals when the ST10F276 reads data from these external devices. During reset, during Hold mode and during Adapt mode an internal pull-up ensures an inactive high level on the $\overline{RD}$ output.
$\overline{WR}$/$\overline{WRL}$	**External Write/Write Low Strobe:** controls the data transfer from the ST10F276 to an external memory or peripheral device. This pin may either provide a general $\overline{WR}$ signal activated for both byte and word write accesses, or specifically control the low byte of an external 16-bit device ($\overline{WRL}$) together with the signal $\overline{WRH}$ (alternate function of P3.12/$\overline{BHE}$). During reset, during Hold mode and during Adapt mode an internal pull-up ensures an inactive (high) level on the $\overline{WR}$/$\overline{WRL}$ output.
$\overline{READY}$/READY	**Ready Input:** receives a control signal from an external memory or peripheral device that is used to terminate an external bus cycle, provided that this function is enabled for the current bus cycle. $\overline{READY}$/READY may be used as synchronous $\overline{READY}$/READY or may be evaluated asynchronously. The polarity can be set to $\overline{READY}$ or READY by setting bit 13 in the BUSCON register.
$\overline{EA}$ / V_{STBY}	**External Access Enable:** determines, if the ST10F276 after reset starts fetching code from the internal Memory area ($\overline{EA}$ = '1') or via the external bus interface ($\overline{EA}$ = '0'). This pin is also used (when Stand-by mode is entered, that is ST10F276 under reset and main V_{DD} turned off) to bias the 32 kHz oscillator amplifier circuit and to provide a reference voltage for the low-power embedded voltage regulator, which generates the internal 1.8V supply for the RTC module (when not disabled) and to retain data inside the Stand-by portion of the XRAM (16 Kbyte). It can range from 4.5 to 5.5V (6V for a reduced amount of time during the device life, 4.0V when RTC and 32 kHz on-chip oscillator amplifier are turned off). In running mode, this pin can be tied low during reset without affecting 32 kHz oscillator, RTC and XRAM activities, since the presence of a stable V_{DD} guarantees the proper biasing of all those modules.
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 ST10F276 into Power Down mode.

Table 29. Summary of dedicated pins (continued)

Pin(s)	Function
RSTIN	**Reset Input:** puts the ST10F276 into the reset default configuration either at Power-On or external events like a hardware failure or manual reset. The input circuitry of the $\overline{\text{RSTIN}}$ pin implements an analog filter in order to minimize the noise sensitivity of the reset input.
RSTOUT	**Reset Output:** provides a special reset signal for external circuitry. $\overline{\text{RSTOUT}}$ is activated at the beginning of the reset sequence, triggered via $\overline{\text{RSTIN}}$, a watchdog timer overflow or by the SRST instruction. $\overline{\text{RSTOUT}}$ remains active (low) until the EINIT instruction is executed. This allows to initialize the controller before the external circuitry is activated.
XTAL1, XTAL2	**Main 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 (only when Direct Drive configuration is selected).
XTAL3, XTAL4	**32 kHz Oscillator Input/Output:** connect the 32 kHz internal clock oscillator to the external crystal. When not used, the XTAL3 should be tied to ground to avoid spurious consumption, while XTAL4 should be left unconnected; besides, bit OFF32 in RTCCON register should be set. 32 kHz oscillator can only be driven by an external crystal, and not by a different external clock source.
V_{AREF} , V_{AGND}	**ADC Reference Supplies**: provide the reference voltage and ground for the Analog to Digital Converter. Besides, these pins provide also the power supply to analog circuitry of the ADC module itself.
V_{DD} , V_{SS}	**Digital Power Supply and Ground:** provide the power supply for the digital logic of the ST10F276. All V_{DD} (8) pins and all V_{SS} (9) pins must be connected to the power supply and ground, respectively.
RPD	**Exit from Power Down:** If a Fast External Interrupt pin (EX7IN...EX0IN) is used to exit from Power Down mode, an external RC circuit should be connected to the RPD pin. The discharging of the external capacitor causes a delay that allows the oscillator and PLL circuits to stabilize before the clock signal is delivered to the CPU and peripherals (see *Figure 56*). For more information on exiting Power Down mode refer to *Section 24: Power reduction modes on page 502*.
V_{18}	**1.8V Decoupling pin:** a decoupling capacitor must be connected between this pin and nearest V_{SS} pin.

Figure 56. RPD external RC circuit



Note: RPD external RC circuit is used for exiting Power Down mode with external interrupt and for Power-on with an asynchronous reset.

8 The external bus interface

The on-chip peripherals and the on-chip RAM and Flash Memory only cover a small fraction of the ST10F276 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.

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

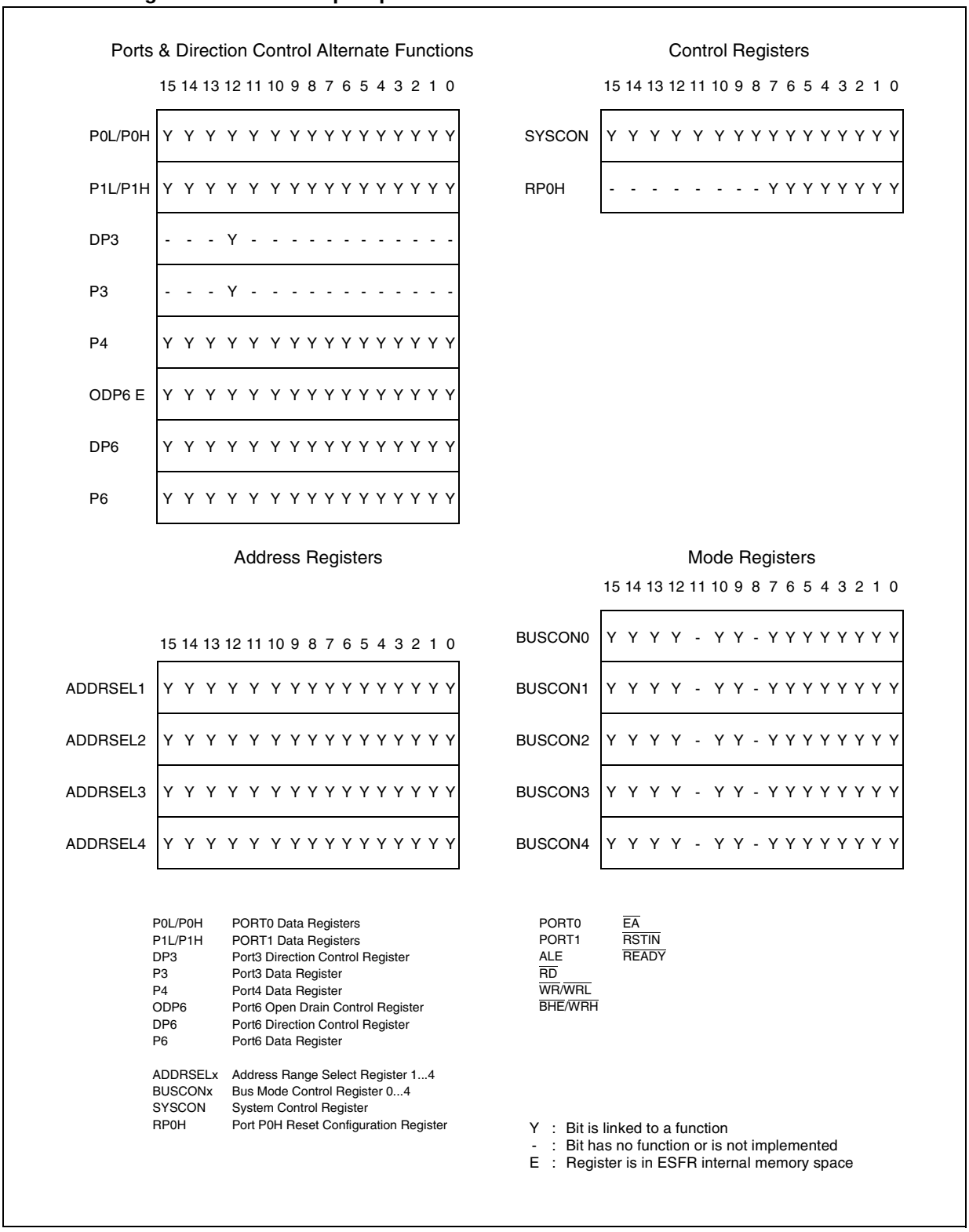
Single chip mode is entered, when pin $\overline{\text{EA}}$ is high during reset. In this case register BUSCON0 is initialized with 0000h, which also resets bit BUSACT0, so no external bus is enabled.

In single chip mode the ST10F276 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 ST10F276 to start execution out of the internal Flash program memory.

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

Figure 57. 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' of BUSCONx register) and configured (bit-field BTYP), the ST10F276 uses a subset of its port lines together with some control lines to build the external bus.

BTYP encoding	External data bus width	External address bus mode
0 0	8-bit Data	De-multiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	De-multiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

The bus configuration (BTYP) for the address windows (BUSCON4...BUSCON1) is selected by software, usually during the initialization of the system.

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

The 16 Mbyte address space of the ST10F276 is divided into 256 segments of 64 Kbytes each. The 16-bit intra-segment address is output on PORT0 for multiplexed bus modes or on PORT1 for de-multiplexed bus modes.

When segmentation is disabled, only one 64 Kbyte segment can be used and accessed. Otherwise, additional address lines may be output on Port4, and/or several chip select lines may be used to select different memory banks or peripherals. These functions are selected during reset via bit-fields SALSEL and CSSEL of register RP0H, respectively.

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

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.

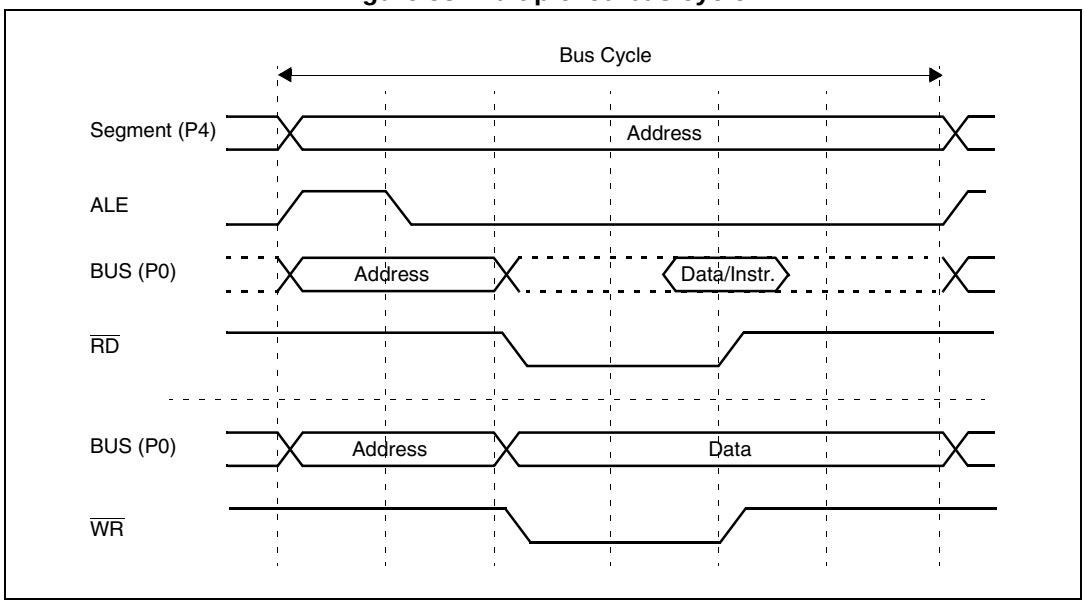
The EBC initiates an external access by generating the Address Latch Enable signal (ALE) and then placing an address on the bus. The falling edge of ALE triggers an external latch to capture the address.

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 de-activated. 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 58. Multiplexed bus cycle



8.2.2 De-multiplexed bus modes

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

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

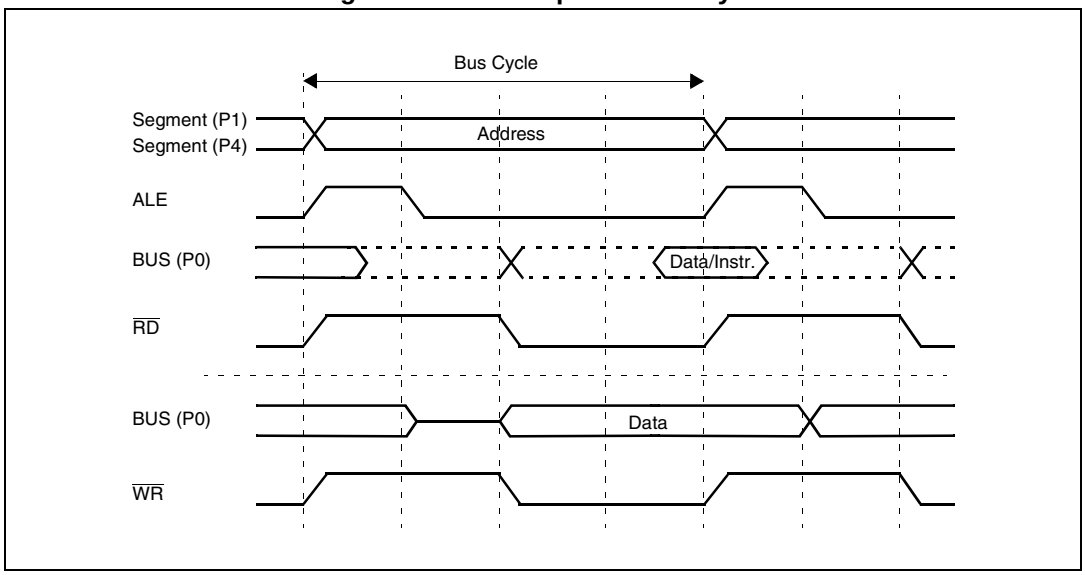
The EBC initiates an external access by placing an address on the address bus. After a programmable period of time the EBC activates the respective command signal ($\overline{RD}$, $\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$).

Data is driven onto the data bus, either by the EBC (for write cycles), or by the external memory/peripheral (for read cycles). After a period of time which is determined by the access time of the memory/peripheral, data become valid.

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

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

Figure 59. De-multiplexed 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 de-multiplexed buses or use READY control or predefined wait-states.

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

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

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

Note: Never change the configuration for an address area that currently supplies the instruction stream. Due to the internal pipeline 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, if the access is made internally, uses one of the address windows defined by ADDRSEL4...1, or

uses the default configuration in BUSCON0. After initializing the active registers, they are selected and evaluated automatically by interpreting the physical address. No additional switching or selecting is necessary during run time, except when more than the four address windows plus the default is to be used.

Switching from de-multiplexed to multiplexed bus mode represents a special case. The bus cycle is started by activating ALE and driving the address to Port4 and PORT1 as usual, if another BUSCON register selects a de-multiplexed bus. However, in the multiplexed bus modes the address is also required on PORT0. In this special case the address on PORT0 is delayed by one CPU clock cycle, which delays the complete (multiplexed) bus cycle and extends the corresponding ALE signal (see *Figure 60*).

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

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.

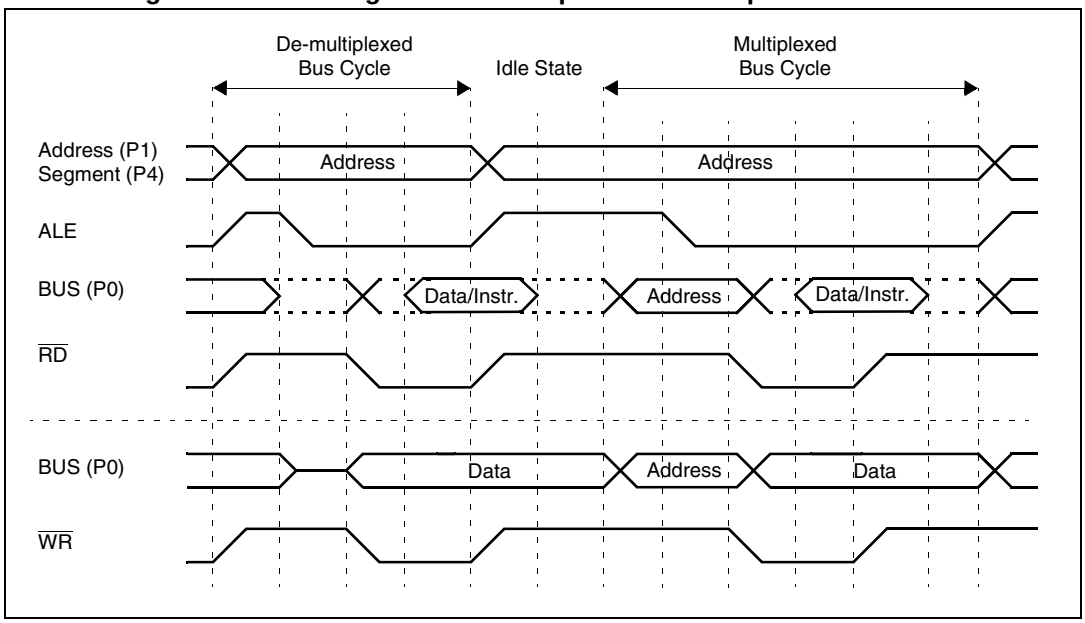
Word accesses on an 8-bit data bus are automatically split into two subsequent byte accesses, where the low byte is accessed first, then the high byte. The assembly of byte to words and the disassembly of words into byte is handled by the EBC and is transparent to the CPU and the programmer.

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 $\overline{BHE}$ signal, while the lower byte is selected with the A0 signal. So the two bytes of the memory can be enabled independent from each other, or together when accessing words.

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

When reading byte from an external 16-bit device, whole words may be read and the ST10F276 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, such as FIFOs or interrupt status registers. In this case individual byte should be selected using $\overline{BHE}$ and A0.

Figure 60. Switching from de-multiplexed to multiplexed bus mode



Bus Mode	Transfer Rate (Speed factor for byte/word/Dword access)	System Requirements	Free I/O Lines
8-bit multiplexed	Very low (1.5 / 3 / 6)	Low (8-bit latch, byte bus)	P1H, P1L
8-bit de-multiplexed	Low (1 / 2 / 4)	Very low (no latch, byte bus)	P0H
16-bit multiplexed	High (1.5 / 1.5 / 3)	High (16-bit latch, word bus)	P1H, P1L
16-bit de-multiplexed	Very high (1 / 1 / 2)	Low (no latch, word bus)	---

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

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

Bit BYTDIS is provided for controlling the active low Byte High Enable ($\overline{\text{BHE}}$) pin. The function of the $\overline{\text{BHE}}$ pin is enabled, if the BYTDIS bit contains a '0'. Otherwise, it is disabled and the pin can be used as standard I/O pin. The $\overline{\text{BHE}}$ pin is implicitly used by the External Bus Controller to select one of two byte-organized memory chips, which are connected to the ST10F276 via a word-wide external data bus. After reset the $\overline{\text{BHE}}$ function is automatically enabled (BYTDIS = '0'), if a 16-bit data bus is selected during reset, otherwise it is disabled (BYTDIS = ‘1’). It may be disabled, if byte access to 16-bit memory is not required, and the $\overline{\text{BHE}}$ signal is not used.

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 RP0H (see table below).

SALSEL	Segment address lines	Directly accessible address space
1 1	Two: A17...A16	256 Kbytes (Default without pull-downs on P0)
1 0	Eight: A23...A16	16 Mbytes (Maximum)
0 1	None	64 Kbytes (Minimum)
0 0	Four: A19...A16	1 Mbyte

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

8.2.7 $\overline{CS}$ signal generation

During external accesses the EBC can generate a (programmable) number of $\overline{CS}$ lines on Port6, which allows to directly select external peripherals or memory banks without requiring an external decoder. The number of $\overline{CS}$ lines is selected during reset and coded in bit field CSSEL in register RP0H (see table below)

CSSEL	Chip select lines	Note
1 1	Five: $\overline{CS4}$...$\overline{CS0}$	Default without pull-downs on P0
1 0	None	Port6 pins free for I/O
0 1	Two: $\overline{CS1}$...$\overline{CS0}$	
0 0	Three: $\overline{CS2}$...$\overline{CS0}$	

The $\overline{CS}$ outputs are associated with the BUSCONx registers and are driven active (low) for any access within the address area defined for the respective BUSCON register.

For any access outside this defined address area the respective $\overline{CS}$ signal will go inactive (high). At the beginning of each external bus cycle the corresponding valid $\overline{CS}$ signal is determined and activated. All other $\overline{CS}$ lines are deactivated (driven high) at the same time.

Note: The $\overline{CS}$ signals will not be updated for an access to any internal address area (for example when no external bus cycle is started), even if this area is covered by the respective ADDRSELx register. An access to an on-chip X-Peripheral deactivates all external $\overline{CS}$ signals. Upon accesses to address windows without a selected $\overline{CS}$ line all selected $\overline{CS}$ lines are deactivated.

The chip select signals allow to operate in four different modes, which are selected via bit CSWENx and CSRENx in the respective BUSCONx register.

CSWENx	CSRENx	Chip select mode
0	0	Address Chip Select (Default after Reset, mode for $\overline{CS0}$)
0	1	Read Chip Select

CSWENx	CSRENx	Chip select mode
1	0	Write Chip Select
1	1	Read/Write Chip Select

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

Read or write chip select signals remain active only as long as the associated control signal ($\overline{RD}$ or $\overline{WR}$) is active.

This also includes the programmable read/write delay. Read chip select is only activated for read cycles, write chip select is only activated for write cycles, read/write chip select is activated for both read and write cycles (write cycles are assumed, if any of the signals $\overline{WRH}$ or $\overline{WRL}$ 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

The external bus interface supports many configurations for the external memory. By increasing the number of segment address lines, a linear address space of 256 Kbytes, 1 Mbyte or 16 Mbytes can be addressed.

It is possible to implement a large memory area and to access a great number of external devices using an external decoder. By increasing the number of $\overline{CS}$ lines, accesses can be made to memory banks or peripherals without external glue logic.

These two features may be combined to optimize the overall system performance. Enabling 4 segment address lines and 5 chip select lines to give access to five memory banks of 1 Mbyte each, so the available address space is 5 Mbytes (without glue logic).

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

8.3 Programmable bus characteristics

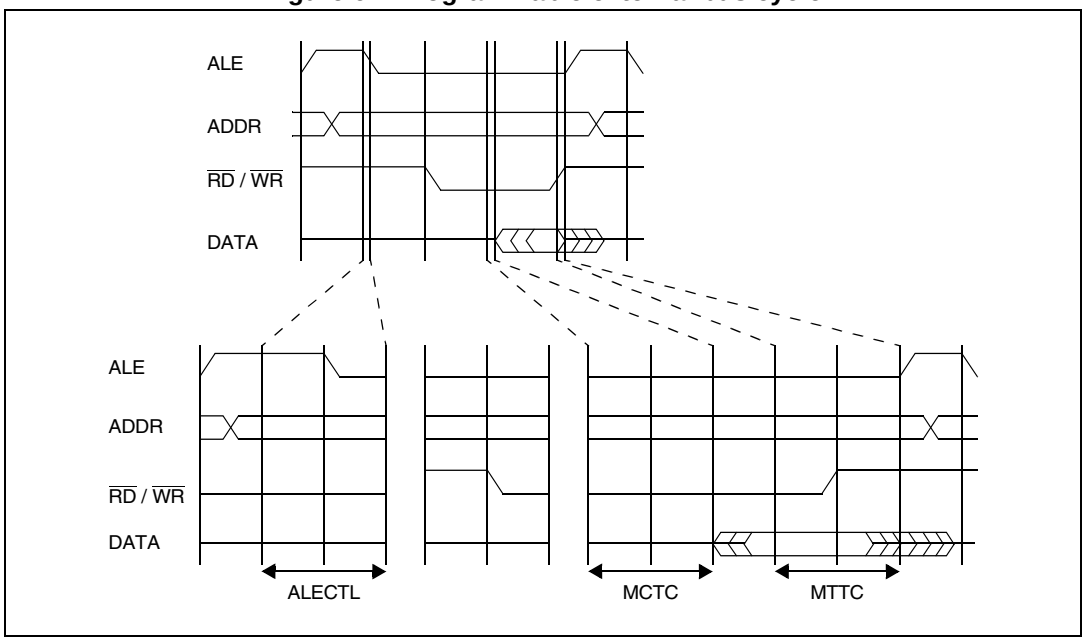
Important timing characteristics of the external bus interface have been made user programmable to allow to adapt it to a wide range of different external bus and memory configurations with different types of memories and/or peripherals.

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

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

Figure 61. Programmable external bus cycle



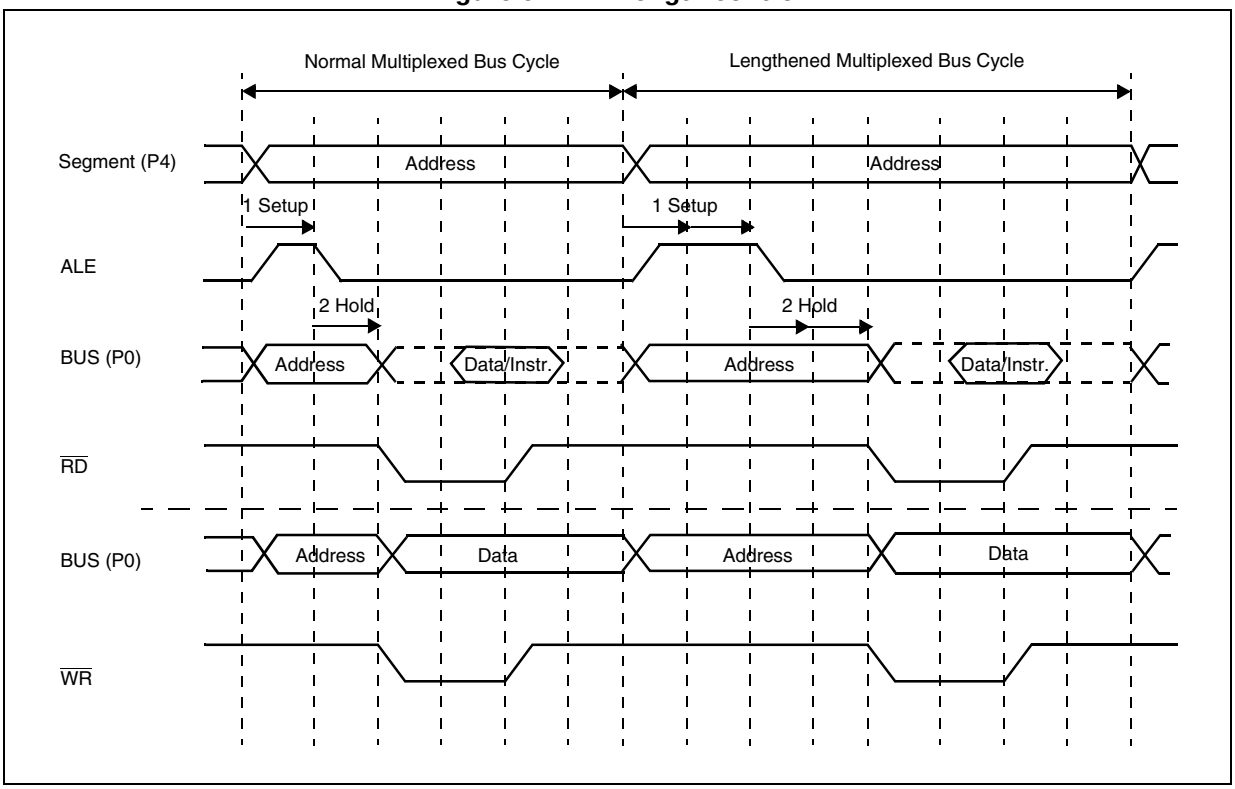
8.3.1 ALE length control

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 62. ALE length control

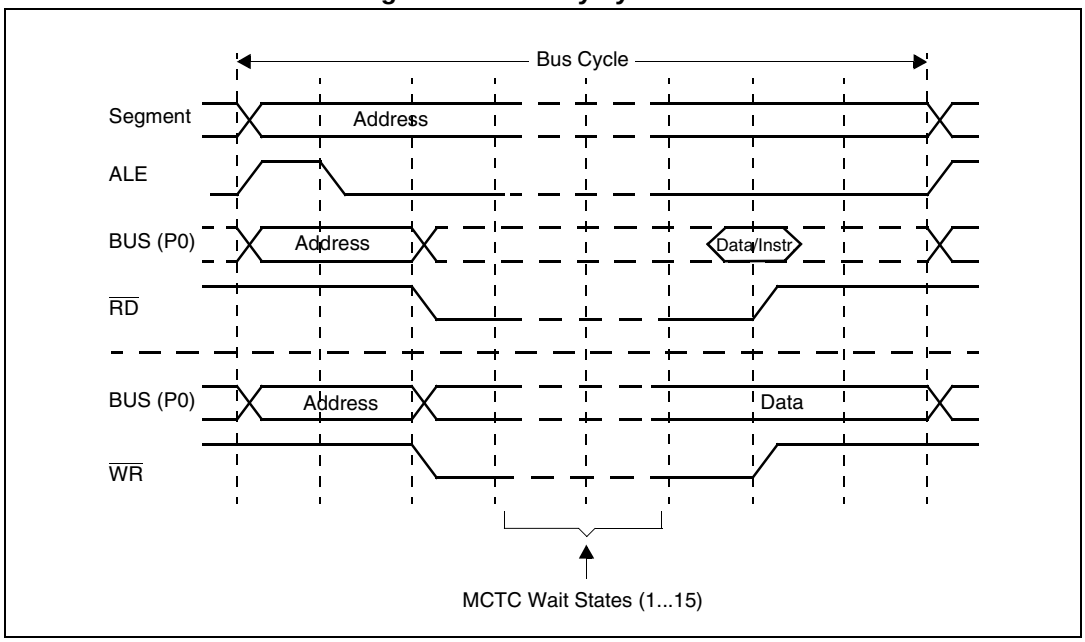


8.3.2 Programmable memory cycle time

The ST10F276 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 ST10F276 can be extended for a memory or a peripheral which cannot keep pace with the controller's maximum speed: some wait-states are introduced during the access (see *Figure 63*). During these memory cycle time wait-states, the CPU is idle, if this access is required for the execution of the current instruction. The memory cycle time wait-states can be programmed in increments of one CPU clock within a range from 0 to 15 (default after reset) via the MCTC fields of the BUSCON registers. 15-[MCTC] wait-states will be inserted.

Figure 63. Memory cycle time



8.3.3 Programmable memory tri-state time

The ST10F276 allows the user to adjust the time between two subsequent external accesses to address slow external devices. The tri-state time MTTC starts when the external device has released the bus after deactivation of the read command ($\overline{RD}$).

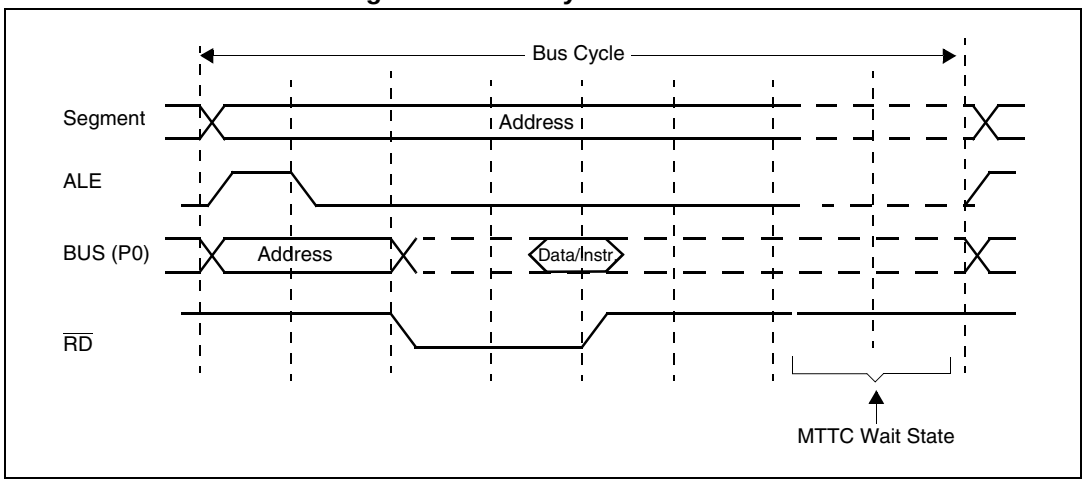
The output of the next address on the external bus can be delayed for a memory or peripheral which needs more time to switch off its bus drivers, by introducing a wait-state after the previous bus cycle (see *Figure 64*).

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

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

External bus cycles in multiplexed bus modes implicitly add one tri-state time wait-state in addition to the programmable MTTC wait-state.
Any MTTC wait-states are applicable to both read and write cycles.

Figure 64. Memory tri-state time



8.3.4 Read / write signal delay

The ST10F276 allows the user to adjust the timing of the read and write commands to account for timing requirements of external peripherals.

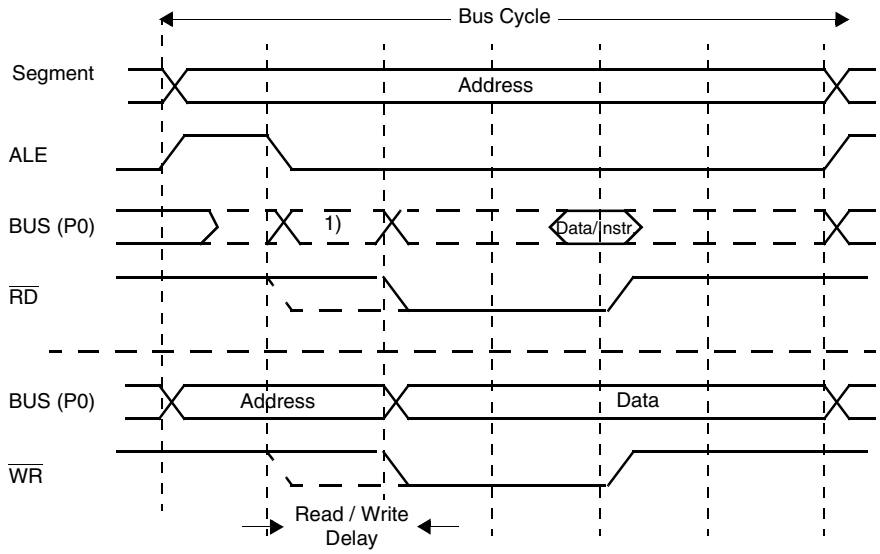
The read/write delay controls the time between the falling edge of ALE and the falling edge of the command. Without read/write delay the falling edges of ALE and command(s) are coincident (except for propagation delays). With the delay enabled, the command(s) become active half a CPU clock cycle after 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 ST10F276's address, when the early $\overline{RD}$ signal is used. Therefore multiplexed bus cycles should always be programmed with read/write delay.

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

Figure 65. Read / write delay



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

8.3.5 READY polarity

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

8.3.6 READY / $\overline{READY}$ controlled bus cycles

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

For situations where the programmable wait-states are not enough, or where the response (access) time of a peripheral is not constant, the ST10F276 provides external bus cycles that are terminated by a READY or $\overline{READY}$ input signal (synchronous or asynchronous). In this case the ST10F276 first inserts a programmable number of wait-states (0...7) and then monitors the READY or $\overline{READY}$ line to determine the actual end of the current bus cycle. The external device drives READY or $\overline{READY}$ low in order to indicate that data have been latched (write cycle) or are available (read cycle).

When the READY or $\overline{READY}$ 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 66*).

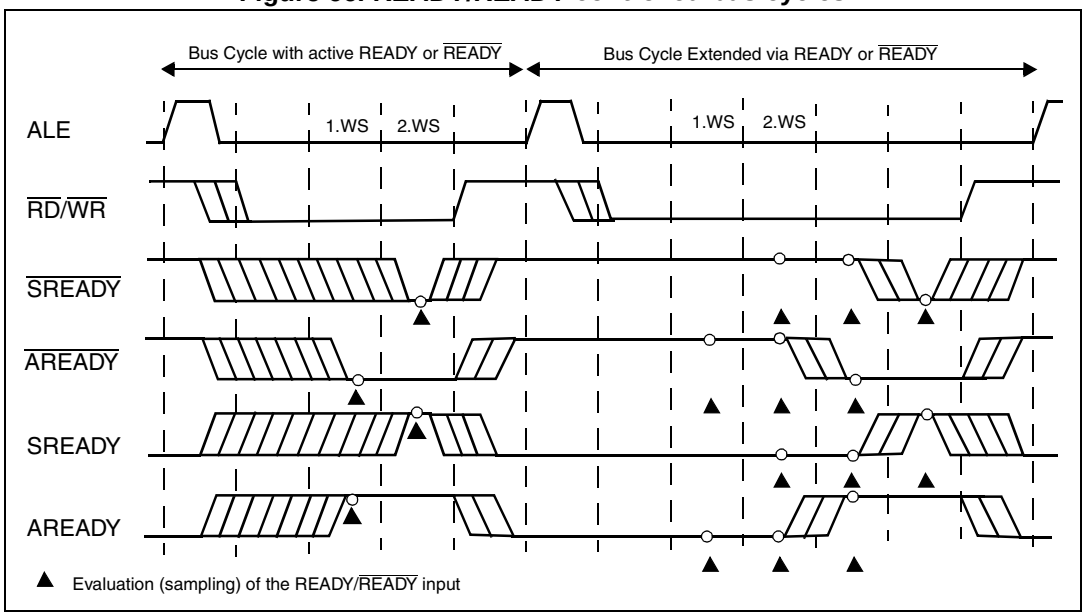
The READY/$\overline{READY}$ function is enabled by the RDYENx bit in the BUSCON registers. When this function is selected (RDYENx = '1'), only the lower 3 bits of the respective MCTC bit-field define the number of inserted wait-states (0...7), while the MSB of bit-field MCTC selects the $\overline{READY}$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

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

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

The asynchronous READY/$\overline{\text{READY}}$ (AREADY / $\overline{\text{AREADY}}$) is less restrictive, but requires additional wait-states caused by the internal synchronization. As the asynchronous READY/$\overline{\text{READY}}$ is sampled earlier (see *Figure 66*) programmed wait-states may be necessary to provide proper bus cycles (see also notes on “normally-ready” peripherals below).

Figure 66. READY/$\overline{\text{READY}}$ controlled bus cycles



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

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

Combining the READY function with predefined wait-states is advantageous in two cases:

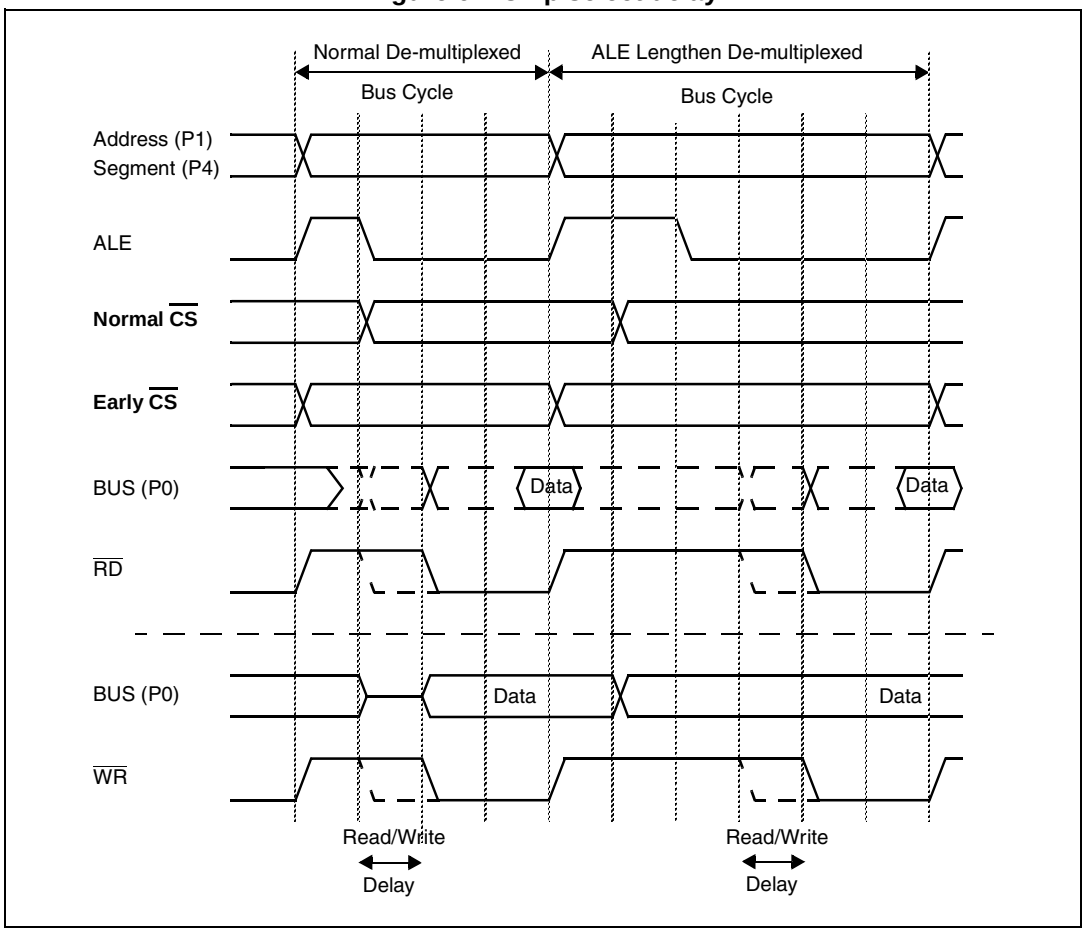
- Memory components with a fixed access time and peripherals operating with READY/$\overline{\text{READY}}$ may be grouped into the same address window. The (external) wait-state control logic in this case would activate READY/$\overline{\text{READY}}$ either upon the memory’s chip select or with the peripheral’s READY/$\overline{\text{READY}}$ output. After the predefined number of wait-states the ST10F276 will check its READY/$\overline{\text{READY}}$ line to determine the end of the bus cycle. For a memory access it will be low already (see *Figure 66*), for a peripheral access it may be delayed. As memories tend to be faster than peripherals, there should be no impact on system performance.
- When using the READY/$\overline{\text{READY}}$ function with so-called “normally-ready” peripherals, it may lead to erroneous bus cycles, if the READY/$\overline{\text{READY}}$ line is sampled too early. These peripherals pull their READY/$\overline{\text{READY}}$ output low, while they are idle. When they are accessed, they deactivate READY/$\overline{\text{READY}}$ until the bus cycle is complete, then

drive it low again. If, however, the peripheral deactivates READY/$\overline{\text{READY}}$ after the first sample point of the ST10F276, the controller samples an active READY/$\overline{\text{READY}}$ and terminates the current bus cycle, which, of course, is too early. By inserting predefined wait-states the first READY/$\overline{\text{READY}}$ sample point can be shifted to a time, where the peripheral has safely controlled the READY/$\overline{\text{READY}}$ line (after two wait-states in the *Figure 66*).

8.3.7 Programmable chip select timing control

The position of the $\overline{\text{CS}}$ lines can be changed. By default (after reset), the $\overline{\text{CS}}$ lines change half a CPU clock cycle after the rising edge of ALE. With the CSCFG bit set in the SYSCON register, the $\overline{\text{CS}}$ lines change with the rising edge of ALE, therefore the $\overline{\text{CS}}$ lines change at the same time that the address lines are changed.

Figure 67. 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 ($\overline{\text{WR}}$, $\overline{\text{BHE}}$), segmentation and internal Memory mapping are controlled by the SYSCON register.

The properties of a bus cycle like chip select mode, usage of $\overline{\text{READY}}$, length of ALE, external bus mode, read/write delay and wait-states are controlled by BUSCON4...BUSCON0 registers. Four of these registers (BUSCON4...BUSCON1) have an associated address select register (ADDRSEL4...ADDRSEL1) which allows to specify up to four address areas and the individual bus characteristics within these areas. All accesses that are not covered by these four areas are then controlled via BUSCON0. This allows to use memory components or peripherals with different interfaces within the same system, while optimizing accesses to each of them.

BUSCON4...BUSCON0 bit SGTDIS controls the correct stack operation (push/pop of CSP or not) during traps and interrupts.

SYSCON (FF12h / 89h) SFR Reset Value: 0xx0h

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

Reset Value: 0000 0xx0 x000 0000b

Bit	Function
XPER-SHARE	**XBUS Peripheral Share Mode Control** ‘0’: External accesses to XBUS peripherals are disabled. ‘1’: XRAM1 and XRAM2 are accessible via the external bus during hold mode. External accesses to the other XBUS peripherals are not guaranteed in terms of AC timings. See *Section 2.4.1: XRAM access via external masters on page 50* for additional details.
VISIBLE	**Visible Mode Control** ‘0’: Accesses to XBUS peripherals are done internally. ‘1’: XBUS peripheral accesses are made visible on the external pins.
XPEN	**XBUS Peripheral Enable bit** ‘0’: Accesses to the on-chip X-peripherals and XRAM are disabled. ‘1’: The on-chip X-peripherals are enabled.
BDRSTEN	**Bidirectional Reset Enable** ‘0’: $\overline{\text{RSTIN}}$ 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 internal reset sequence.
OWDDIS	**Oscillator Watchdog Disable Control** ‘0’: Oscillator Watchdog (OWD) is enabled. If PLL is bypassed, the OWD monitors XTAL1 activity. If there is no activity on XTAL1 for at least 1 µs, the CPU clock is switched automatically to PLL’s base frequency (from 750 kHz to 3 MHz). ‘1’: OWD is disabled. If the PLL is bypassed, the CPU clock is always driven by XTAL1 signal. The PLL is turned off to reduce power supply current.
PWDCFG	**Power Down Mode Configuration Control** ‘0’: Power Down Mode can only be entered during PWRDN instruction execution if $\overline{\text{NMI}}$ pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must 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 or with external reset.

Bit	Function
CSCFG	**Chip Select Configuration Control** '0': Latched Chip Select lines, $\overline{CSx}$ changes 1 TCL after rising edge of ALE. '1': Unlatched Chip Select lines, $\overline{CSx}$ changes with rising edge of ALE.
WRCFG	**Write Configuration Control** (Inverted copy of WRC bit of RP0H) '0': Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function. '1': Pin $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$.
CLKEN	**System Clock Output Enable** (CLKOUT) '0': CLKOUT disabled, pin may be used for general purpose I/O. '1': CLKOUT enabled, pin outputs the system clock signal or a prescaled value of system clock according to XCLKOUTDIV register setting.
BYTDIS	**Disable/Enable Control for Pin BHE** (Set according to data bus width) '0': Pin $\overline{BHE}$ enabled. '1': Pin $\overline{BHE}$ disabled, pin may be used for general purpose I/O.
ROMEN	**Internal Memory Enable** (Set according to pin $\overline{EA}$ during reset) '0': Internal memory disabled: accesses to the IFlash Memory area use the external bus. '1': Internal memory enabled.
SGTDIS	**Segmentation Disable/Enable Control** '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit). '1': Segmentation disabled (Only IP is saved/restored).
ROMS1	**Internal Memory Mapping** '0': Internal memory area mapped to segment 0 (00'0000h...00'7FFFh). '1': Internal memory area mapped to segment 1 (01'0000h...01'7FFFh).
STKSZ	**System Stack Size** Selects the size of the system stack (in the IRAM) from 32 to 1024 words.

The layout of the five BUSCON registers is identical. Registers BUSCON4...BUSCON1, which control the selected address windows, are completely under software control, while register BUSCON0, which is also used for the very first code access after reset, is partly controlled by hardware, and it is initialized via PORT0 during the reset sequence.

This hardware control allows to define an appropriate external bus for systems, where no internal program memory is provided.

BUSCON0 (FF0Ch / 86h) SFR Reset Value: 0xx0h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN0	CSREN0	RDYPOL0	RDYEN0	-	**BUSACT0**	ALECTL0	-	**BTYP**		MTTC0	RWDC0	MCTC			
RW	RW		RW		RW	RW		RW		RW	RW	RW			

BUSCON0 Reset Value: *0000 0xx0 xx00 0000b*

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN1	CSREN1	RDYPOL1	RDYEN1	-	BUSACT1	ALECTL1	-	BTYP		MTTC1	RWDC1	MCTC			
RW	RW	RW	RW		RW	RW		RW		RW	RW	RW			

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN2	CSREN2	RDYPOL2	RDYEN2	-	BUSACT2	ALECTL2	-	BTYP		MTTC2	RWDC2	MCTC			
RW	RW	RW		RW	RW		RW			RW		RW			

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			

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		MTiTC4	RWDC4	MCTC			
RW	RW	RW		RW	RW		RW			RW		RW			

Bit	Function
MCTC	**Memory Cycle Time Control** (Number of memory cycle time wait-states) ‘0000’: 15 wait-states (Number of wait-states = 15 - [MCTC]). ‘1111’: No wait-states.
RWDCx	**Read/Write Delay Control for BUSCONx** ‘0’: With read/write delay, the CPU inserts 1 TCL after falling edge of ALE. ‘1’: No read/write delay, $\overline{RW}$ is activated after falling edge of ALE.
MTTCx	**Memory Tristate Time Control** ‘0’: 1 wait-state. ‘1’: No wait-state.
BTYP	**External Bus Configuration** ‘00’: 8-bit De-multiplexed Bus ‘01’: 8-bit Multiplexed Bus ‘10’: 16-bit De-multiplexed Bus ‘11’: 16-bit Multiplexed Bus **Note:** For BUSCON0 BTYP is defined via PORT0 during reset.
ALECTLx	**ALE Lengthening Control** ‘0’: Normal ALE signal. ‘1’: Lengthened ALE signal.
BUSACTx	**Bus Active Control** ‘0’: External bus disabled. ‘1’: External bus enabled (within the respective address window, see ADDRSEL).
RDYENx	**READY Input Enable** ‘0’: External bus cycle is controlled by bit field MCTC only. ‘1’: External bus cycle is controlled by the $\overline{READY}$ input signal.
RDYPOLx	**Ready Active Level Control** ‘0’: Active level on the READY pin is low, bus cycle terminates with a ‘0’ on READY pin. ‘1’: Active level on the READY pin is high, bus cycle terminates with a ‘1’ on READY pin.

Bit	Function
CSRENx	**Read Chip Select Enable** '0': The $\overline{CS}$ signal is independent of the read command ($\overline{RD}$). '1': The $\overline{CS}$ signal is generated for the duration of the read command.
CSWENx	**Write Chip Select Enable** '0': The $\overline{CS}$ signal is independent of the write command ($\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$). '1': The $\overline{CS}$ signal is generated for the duration of the write command.

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

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

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

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

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

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

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

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

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

Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ADDRSELx register pair. See *Table 30*.
RGSAD	**Range Start Address** Defines the upper bit of the start address (A23...) of the respective address area. See *Table 30*.

Note: Register BUSCON0 controls the complete external address space, except for the four 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 four separate address areas within the address space of the ST10F276. 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 30*).

For a given window size, only those upper address bits of the start address are used (marked “R”), which are not implicitly used for addresses inside the window. The lower bits of the start address (marked “x”) are disregarded.

Table 30. Definition of address areas

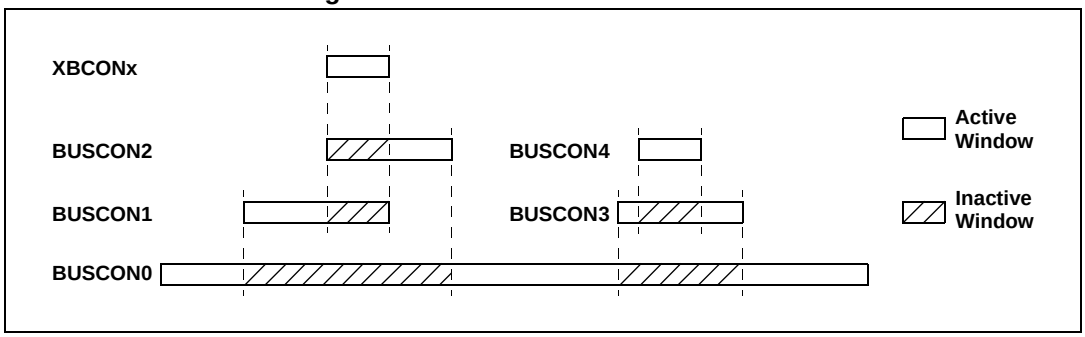
Bit-field RGSZ	Resulting window size	Relevant bit (R) of start address (A23...A12)
		A23 A12
0 0 0 0	4 Kbytes	R R R R R R R R R R R R
0 0 0 1	8 Kbytes	R R R R R R R R R R R x
0 0 1 0	16 Kbytes	R R R R R R R R R R x x
0 0 1 1	32 Kbytes	R R R R R R R R R x x x
0 1 0 0	64 Kbytes	R R R R R R R R x x x x
0 1 0 1	128 Kbytes	R R R R R R R x x x x x
0 1 1 0	256 Kbytes	R R R R R R x x x x x x
0 1 1 1	512 Kbytes	R R R R R x x x x x x x
1 0 0 0	1 Mbyte	R R R R x x x x x x x x
1 0 0 1	2 Mbytes	R R R x x x x x x x x x
1 0 1 0	4 Mbytes	R R x x x x x x x x x x
1 0 1 1	8 Mbytes	R x x x x x x x x x x x
1 1 x x	Reserved	

8.4.2 Address window arbitration

For each access the EBC compares the current address with all address select registers (programmable ADDRSELx and hard-wired XADRSx - Note that XADRS3 is programmable also). This comparison is done in four levels.

- The hard-wired 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 68*).
- A match with registers ADDRSEL1 or ADDRSEL3 directs the access to the respective external area using the corresponding BUSCONx register.
- If there is no match with any XADRSx or ADDRSELx register the access to the external bus uses register BUSCON0.

Figure 68. 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 hard-wired (or programmable) XADRSx registers are defined non-overlapping.

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**CLKCFG**			**SALSEL**		**CSSEL**		**WRC**
								R			R		R		R

Bit	Function
WRC (1)	**Write Configuration Control** (Set according to pin P0H.0 during reset) '0': Pins $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$. '1': Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function.
CSSEL (1)	**Chip Select Line Selection** (Number of active $\overline{CS}$ outputs) 0 0: 3 $\overline{CS}$ lines: $\overline{CS}$2...$\overline{CS}$0 0 1: 2 $\overline{CS}$ lines: $\overline{CS}$1...$\overline{CS}$0 1 0: No $\overline{CS}$ lines at all 1 1: 5 $\overline{CS}$ lines: $\overline{CS}$4...$\overline{CS}$0 (Default without pull-downs)
SALSEL (1)	**Segment Address Line Selection** (Number of active segment address outputs) 0 0: 4-bit segment address: A19...A16 0 1: No segment address lines at all 1 0: 8-bit segment address: A23...A16 1 1: 2-bit segment address: A17...A16 (Default without pull-downs on P0)
CLKCFG (1), (2)	**CPU Frequency**: P0H.7-5 → $f_{CPU} = f_{XTAL} \times F$ 111 → $f_{XTAL} \times 4$ — Default configuration without pull-downs on P0 110 → $f_{XTAL} \times 3$ 101 → $f_{XTAL} \times 8$ 100 → $f_{XTAL} \times 5$ 011 → $f_{XTAL} \times 1$ — Direct drive (3) 010 → $f_{XTAL} \times 10$ 001 → $f_{XTAL} \times 0.5$ — CPU clock via prescaler (3) 000 → $f_{XTAL} \times 16$

1. These bits are set according to Port0 configuration during any reset sequence.
2. RP0H.[7...0] bits are loaded only during a long hardware reset.
3. Refer to datasheet for more details about input clock ranges.

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 de-multiplexed external bus, even for multiplexed bus cycles.
- Not all address areas defined via registers ADDRSELx may overlap each other. The operation of the EBC will be unpredictable in such a case.
- 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) such as IRAM, GPRs or SFRs are used, the external bus interface does not change (see *Table 31*).

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 31*). The address mentioned above includes Port1, Port4, $\overline{BHE}$ and ALE which also pulses for an XBUS cycle. The external $\overline{CS}$ signals on Port6 are driven inactive (high) because the EBC switches to an internal $\overline{XCS}$ signal.

The **external control signals** ($\overline{RD}$ and $\overline{WR}$ or $\overline{WRL}$/$\overline{WRH}$ if enabled) **remain inactive** (high) (see *Table 31*).

Table 31. Status of the external bus interface during EBC idle state

Pins	Internal accesses only	XBUS accesses
PORT0	Tristate (floating)	Tristate (floating) for read accesses XBUS write data for write accesses
PORT1	Last used external address (if used for the bus interface)	Last used XBUS address (if used for the bus interface)
Port4	Last used external segment address (on selected pins)	Last used XBUS segment address (on selected pins)
Port6	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)
$\overline{WR}$/$\overline{WRL}$	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 ST10F276 supports this approach with the possibility to arbitrate the access to its external bus, and to the external devices.

This bus arbitration allows an external master to request the ST10F276's bus via the HOLD input. The ST10F276 acknowledges this request via the HLDA output and will float its bus lines in this case. The CS outputs provide internal pull-up devices.

The new master may now access the peripheral devices or memory banks via the same interface lines as the ST10F276. During this time the ST10F276 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 ST10F276 needs access to its external bus while it is occupied by another bus master, it demands it via the BREQ output.

The external bus arbitration is enabled by setting (to '1') bit HLDEN in register PSW. In this case the three bus arbitration pins HOLD, HLDA and BREQ are automatically controlled by the EBC independent of their I/O configuration. This is not true when XSSC is enabled (setting bit XSSCEN in XPERCON register): the functions HLDA and BREQ are masked, and the related pins are controlled by the XSSCPORT register. Bit HLDEN may be cleared during the execution of program sequences, where the external resources are required but cannot be shared with other bus masters. In this case the ST10F276 will not answer to HOLD requests from other external masters. If HLDEN is cleared while the ST10F276 is in hold state (code execution from IRAM/IFlash): this hold state is left only after HOLD has been deactivated again. In this case the current hold state continues and only the next HOLD request is not answered.

Connecting two ST10F276's in this way would require additional logic to combine the respective output signals HLDA and BREQ. This can be avoided by switching one of the controllers into slave mode where pin HLDA is switched to input.

This allows to directly connect the slave controller to another master controller without glue logic. The slave mode is selected by setting bit DP6.7 to'1'. DP6.7 = '0' (default after reset) selects the Master Mode.

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

8.6.1 Connecting bus masters

When multiple ST10F276's or a ST10F276 and another bus master should share external resources some glue logic is required that defines the currently active bus master and also enables a ST10F276 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 ST10F276 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 ST10F276 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 69*).

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 ST10F276’s external bus is requested by driving its HOLD input low. After synchronizing this signal the ST10F276 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 ST10F276 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 ST10F276 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 2 on page 24*).

Figure 69. Sharing external resources using slave mode

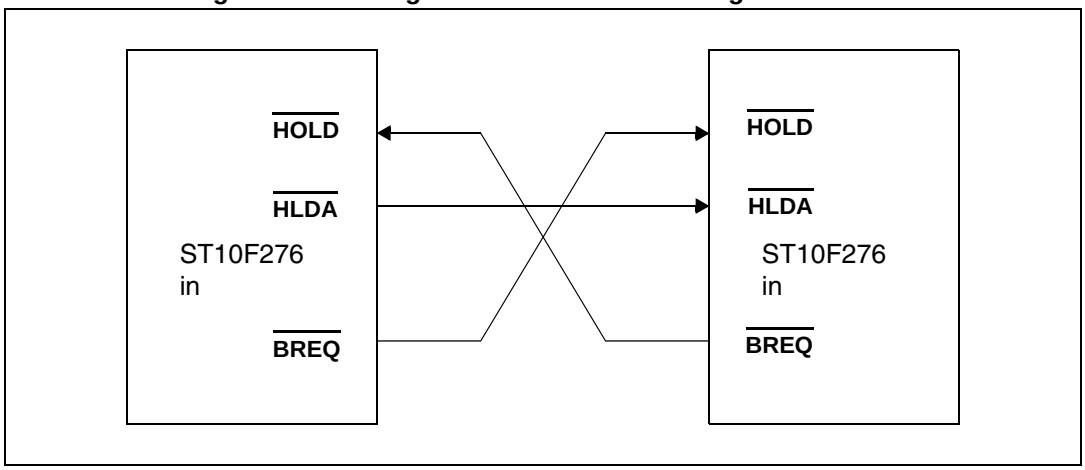
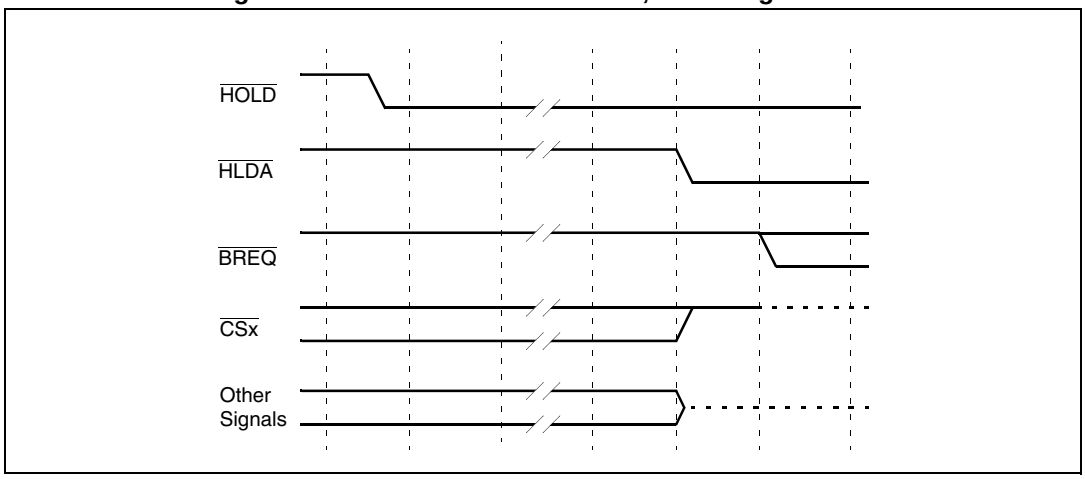


Figure 70. External bus arbitration, releasing the bus



Note: The ST10F276 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.

8.6.3 Exiting the hold state

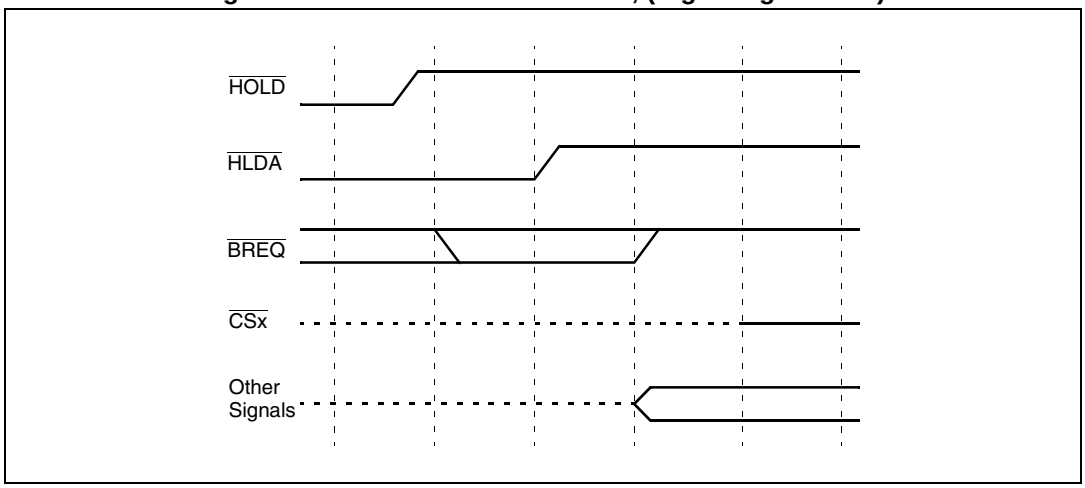
The external bus master returns the access rights to the ST10F276 by driving the HOLD input high. After synchronizing this signal the ST10F276 will drive the HLDA output high, actively drive the control signals and resume executing external bus cycles if required. Depending on the arbitration logic, the external bus can be returned to the ST10F276 under two circumstances:

- The external master does no more require access to the shared resources and gives up its own access rights.
- The ST10F276 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 ST10F276, depending on the priority of the different masters.

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

Figure 71. External bus arbitration, (regaining the bus)



Note: The falling BREQ edge shows the last chance for BREQ to trigger the indicated regain-sequence. Even if BREQ is activated earlier the regain-sequence is initiated by HOLD going high. BREQ and HOLD are connected via an external arbitration circuitry. Note that HOLD may also be deactivated without the ST10F276 requesting the bus.

8.7 The XBUS interface

The ST10F276 provides an on-chip interface (the XBUS interface), which allows to connect integrated costumer / application specific peripherals to the standard controller core.

The XBUS is an internal representation of the external bus interface, it works in the same way.

The current XBUS interface is prepared to support up to three X-Peripherals: for each peripheral on the XBUS (X-Peripheral) there is a separate address window controlled by an XBCON and an XADRS registers.

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.

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.

As the register pairs control integrated peripherals rather than externally connected ones, they are typically fixed by mask programming rather than being user programmable. Nevertheless, on ST10F276, the XADRS3 register is available and programmable for the user. It defines the memory range for XFLASH, Flash Control Registers and XRAM2 accesses.

XADRS3 (F01Ch / 0Eh) ESFR Reset Value: 800Bh

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

Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective XBCONx/XADRSx register pair. See *Table 32*.
RGSAD	**Range Start Address** Defines the upper bit of the start address (A19...) of the respective address area. See *Table 32*.

The register functionality is the same as the one of ADDRSELx registers used for external address range selection. However, it is a protected register and it can only be written before EINIT instruction execution. Note that the range start address can be only on boundaries specified by the selected range size. The following table gives a definition of Range Size Selection and Range Start Address.

Table 32. Definition of XBUS address areas

Bit-field RGSZ	Resulting window size	Relevant bit (R) of start address (A19...A8)
		A19 A8
0 0 0 0	256 bytes	R R R R R R R R R R R
0 0 0 1	512 bytes	R R R R R R R R R R x
0 0 1 0	1 Kbyte	R R R R R R R R R x x
0 0 1 1	2 Kbytes	R R R R R R R R x x x
0 1 0 0	4 Kbytes	R R R R R R R x x x x
0 1 0 1	8 Kbytes	R R R R R R x x x x x
0 1 1 0	16 Kbytes	R R R R R x x x x x x
0 1 1 1	32 Kbytes	R R R R x x x x x x x
1 0 0 0	64 Kbytes	R R R x x x x x x x x
1 0 0 1	128 Kbytes	R R x x x x x x x x x
1 0 1 0	256 Kbytes	R x x x x x x x x x x
1 0 1 1	512 Kbytes	
1 1 x x	Reserved	

Upon Reset, XADRS3 register is programmed so that address range 08'0000h-0F'FFFFh is accessed with the internal XBUS chip select. The range 08'0000h-08'FFFFh is overlapped by IFLASH memory space, which have higher priority on XBUS space.

The address range defined by XADRS3 can be reduced by reprogramming it before EINIT execution: the area which is no longer inside the new address range becomes external memory space (again, apart from range 08'0000h-08'FFFFh dedicated to IFLASH, as long as ROMEN bit in SYSCON register is set). The address range defined by XADRS3 has priority over any external address range defined through ADDRSELx (x = 1...4) registers.

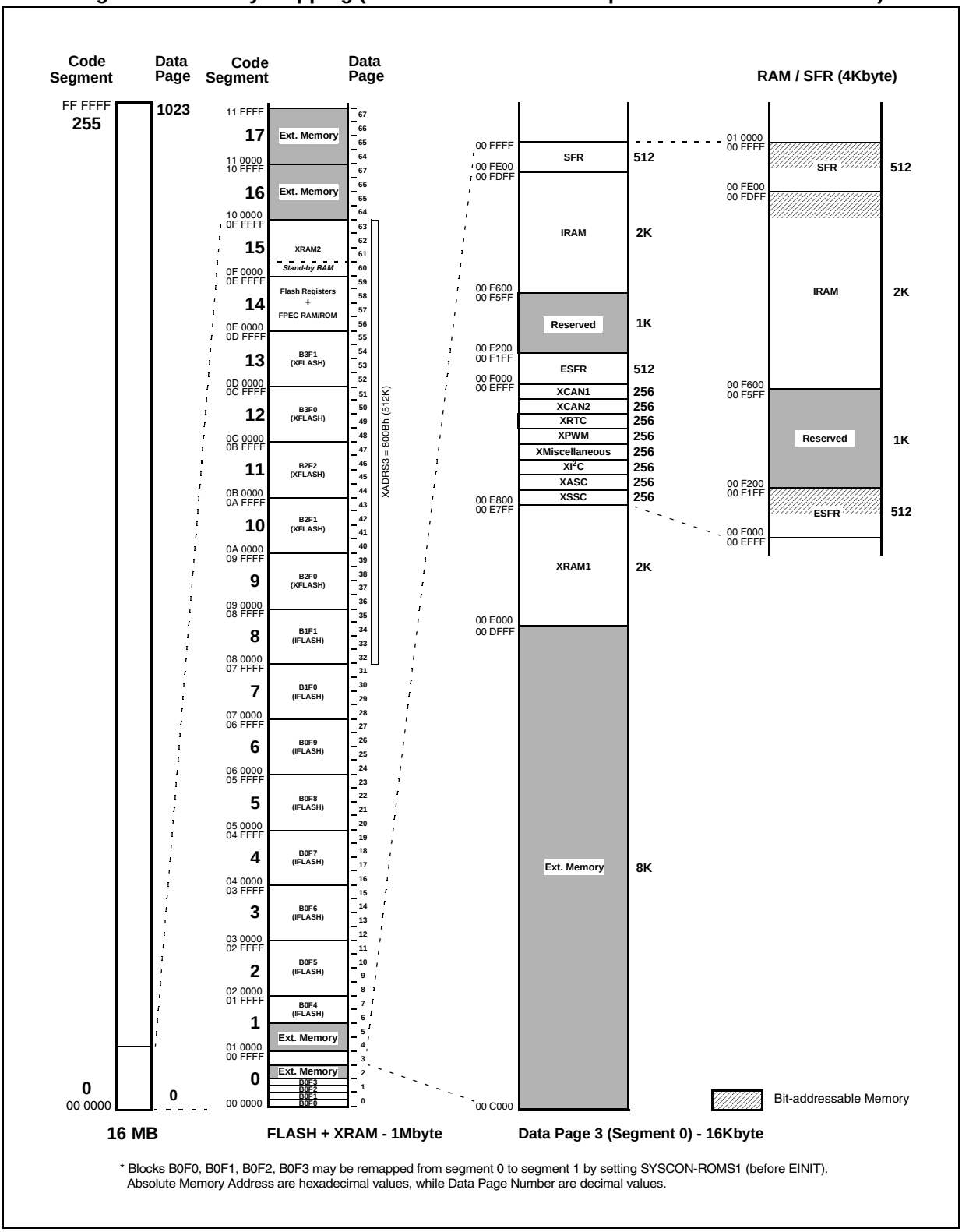
An example of XADRS3 modification with respect to the default (*Figure 72*) value is shown in *Figure 73*: a portion of XFLASH (Bank 2) is no longer available, and all accesses to memory range 09'0000h-0B'FFFFh are redirected to external memory.

Visibility of XBUS peripherals

In order to keep the ST10F276 compatible with the ST10F269, the XBUS peripherals can be selected to be visible and / or accessible on the external address / data bus. Different bits for X-peripheral enabling in XPERCON register must be set. If these bits are cleared before the global enabling with XPEN-bit in SYSCON register, the corresponding address space,

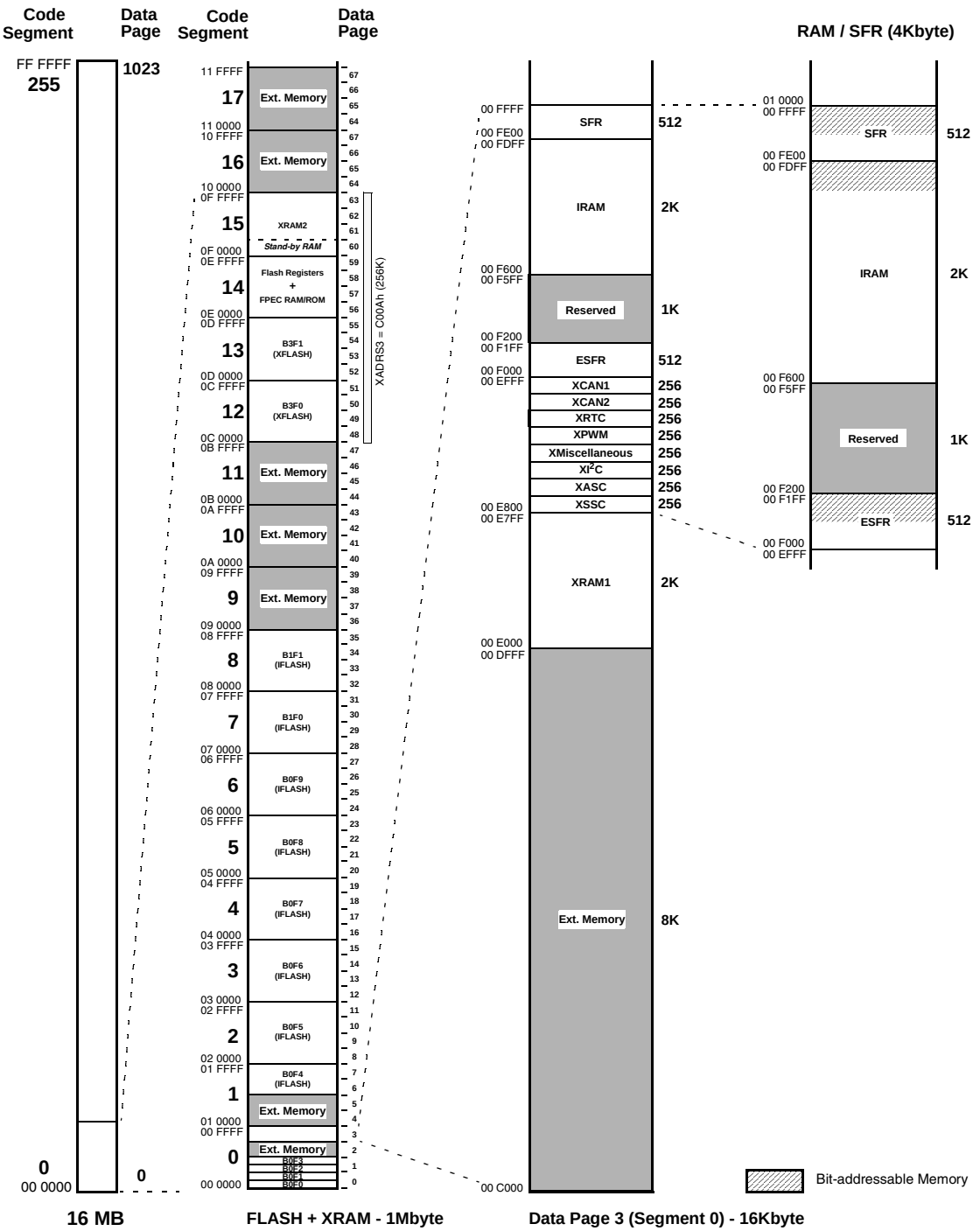
port pins and interrupts are not occupied by the peripheral, the peripheral is then not visible or available. Refer to *Chapter 26: Register set*.

Figure 72. Memory mapping (User mode: Flash read operations / XADRS = 800Bh)



* Blocks B0F0, B0F1, B0F2, B0F3 may be remapped from segment 0 to segment 1 by setting SYSCON-ROMS1 (before EINIT).
Absolute Memory Address are hexadecimal values, while Data Page Number are decimal values.

Figure 73. Memory mapping (User mode: Flash read operations / XADRS = C00Ah)



* Blocks B0F0, B0F1, B0F2, B0F3 may be remapped from segment 0 to segment 1 by setting SYSCON-ROMS1 (before EINIT).
Absolute Memory Address are hexadecimal values, while Data Page Number are decimal values.

XPERCON (F024h / 12h) ESFR Reset Value: - 005h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	XMISC EN	XI2C EN	XSSC EN	XASC EN	XPWM EN	XFLAS HEN	XRTC EN	XRAM2 EN	XRAM1 EN	CAN2 EN	CAN1 EN
-	-	-	-	-	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
CAN1EN	**CAN1 Enable Bit** ‘0’: Accesses to the on-chip CAN1 XPeripheral and its functions are disabled (P4.5 and P4.6 pins can be used as general purpose IOs, but address range 00’EC00h-00’EFFFh is directed to external memory only if CAN2EN, XRTCEN, XASCEN, XSSCEN, XI2CEN, XPWMEN an XMISCEN are ‘0’ also). ‘1’: The on-chip CAN1 XPeripheral is enabled and can be accessed.
CAN2EN	**CAN2 Enable Bit** ‘0’: Accesses to the on-chip CAN2 XPeripheral and its functions are disabled (P4.4 and P4.7 pins can be used as general purpose IOs, but address range 00’EC00h-00’EFFFh is directed to external memory only if CAN1EN, XRTCEN, XASCEN, XSSCEN, XI2CEN, XPWMEN and XMISCEN are ‘0’ also). ‘1’: The on-chip CAN2 XPeripheral is enabled and can be accessed.
XRAM1EN	**XRAM1 Enable Bit** ‘0’: Accesses to the on-chip 2 Kbyte XRAM are disabled. Address range 00’E000h-00’E7FFh is directed to external memory. ‘1’: The on-chip 2 Kbyte XRAM is enabled and can be accessed.
XRAM2EN	**XRAM2 Enable Bit** ‘0’: Accesses to the on-chip 64 Kbyte XRAM are disabled, external access performed. Address range 0F’0000h-0F’FFFFh is directed to external memory only if XFLASHEN is ‘0’ also. ‘1’: The on-chip 64 Kbyte XRAM is enabled and can be accessed.
XRTCEN	**RTC Enable** ‘0’: Accesses to the on-chip RTC module are disabled, external access performed. Address range 00’ED00h-00’EDFF is directed to external memory only if CAN1EN, CAN2EN, XASCEN, XSSCEN, XI2CEN, XPWMEN and XMISCEN are ‘0’ also. ‘1’: The on-chip RTC module is enabled and can be accessed.
XPWMEN	**XPWM Enable** ‘0’: Accesses to the on-chip XPWM module are disabled, external access performed. Address range 00’EC00h-00’ECFF is directed to external memory only if CAN1EN, CAN2EN, XASCEN, XSSCEN, XI2CEN, XRTCEN and XMISCEN are ‘0’ also. ‘1’: The on-chip XPWM module is enabled and can be accessed.
XFLASHEN	**XFlash Enable Bit** ‘0’: Accesses to the on-chip XFlash are disabled, external access performed. Address range 09’0000h-0E’FFFFh is directed to external memory only if XRAM2EN is ‘0’ also. ‘1’: The on-chip XFlash is enabled and can be accessed.
XASCEN	**XASC Enable Bit** ‘0’: Accesses to the on-chip XASC are disabled, external access performed. Address range 00’E900h-00’E9FFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XI2CEN, XPWMEN and XMISCEN are ‘0’ also. ‘1’: The on-chip XASC is enabled and can be accessed.

Bit	Function
XSSCEN	**XSSC Enable Bit** '0': Accesses to the on-chip XSSC are disabled, external access performed. Address range 00'E800h-00'E8FFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XI2CEN, XPWMEN and XMISCEN are '0' also. '1': The on-chip XSSC is enabled and can be accessed.
XI2CEN	**I^2C Enable Bit** '0': Accesses to the on-chip I^2C are disabled, external access performed. Address range 00'EA00h-00'EAFFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XSSCEN, XPWMEN and XMISCEN are '0' also. '1': The on-chip I^2C is enabled and can be accessed.
XMISCEN	**XBUS Additional Features Enable Bit** '0': Accesses to the Additional Miscellaneous Features is disabled. Address range 00'EB00h-00'EBFFh is directed to external memory only if CAN1EN, CAN2EN, XRTCEN, XASCEN, XSSCEN, XPWMEN and XI2CEN are '0' also. '1': The Additional Features are enabled and can be accessed.

Note: 1 When CAN1, CAN2, RTC, XASC, XSSC, I^2C, XPWM and the XBUS Additional Features are all disabled via XPERCON setting, then any access in the address range 00'E800h - 00'EFFFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register. All pins involved with X-Peripherals, can be used as General Purpose I/O whenever the related module is not enabled.

2 The default XPER selection after Reset is identical to XBUS configuration of ST10F168/ST10F269: CAN1 is enabled, CAN2 is disabled, XRAM1 (2 Kbyte compatible XRAM) is enabled, XRAM2 (new 64 Kbyte XRAM) is disabled; all the other X-Peripherals are disabled after Reset.

3 Register XPERCON cannot be changed after the global enabling of X-Peripherals, that is, after setting of bit XPEN in SYSCON register.

4 In Emulation mode, all the X-Peripherals are enabled (XPERCON bits are all set). It is up to the bondout chip to redirect or not an access to external memory or to XBUS.

5 Reserved bits of XPERCON register should be always written to '0'.

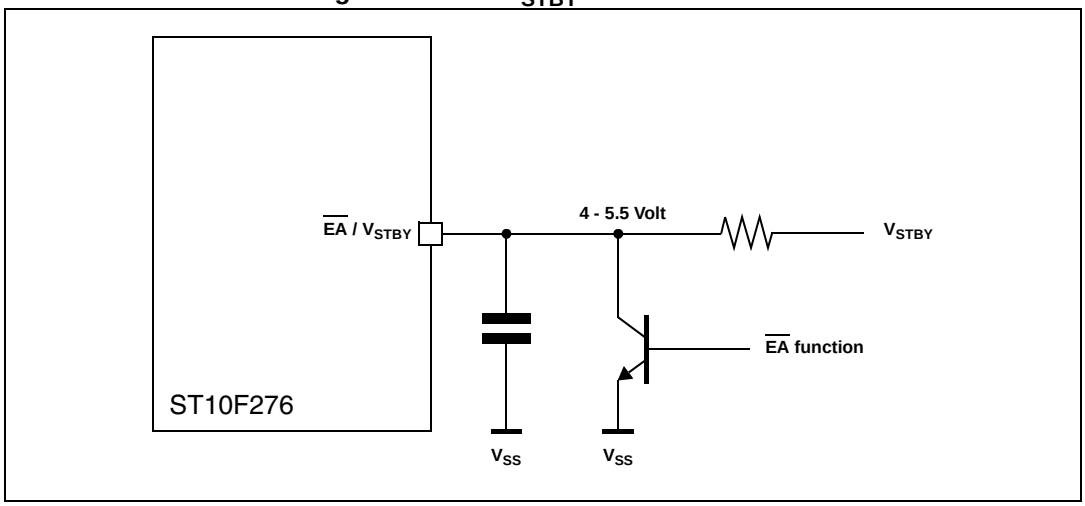
8.8 $\overline{EA}$ functionality

In ST10F276 the $\overline{EA}$ pin is shared with V_{STBY} supply pin. When main V_{DD} is on and stable, V_{STBY} can be temporary grounded: the logic that in Stand-by Mode is powered by V_{STBY} (that is 16 Kbyte portion of XRAM, 32 kHz oscillator, Stand-by Voltage Regulator and Real Time Clock module), is powered by the main V_{DD}. This allows to drive low $\overline{EA}$ pin during reset as requested to configure the system to start from the external memory.

An appropriate external circuit must be provided to manage dynamically both the functionalities associated with the pin: during reset and with stable V_{DD}, the pin can be tied low, while after reset (or anyway before turning off the main V_{DD} to enter in Stand-by mode) the V_{STBY} supply should be applied.

Refer to *Section 24.3: Stand-by mode on page 509* for more details.

Figure 74. $\overline{EA}$ / V_{STBY} external circuit



In *Figure 74* the diagram of a possible external circuit is reported. Attention should be paid in implementing the resistance for current limitation of bipolar: the same resistance should not disturb the Stand-by mode when some current (in the order of hundreds of µA) is provided to the device by the V_{STBY} voltage supply source: the voltage at the pin of ST10F276 should not become lower than 4.5 volt (4.0V when RTC and 32 kHz on-chip oscillator amplifier are turned off).

In order to reduce the effect of the current consumption transients on V_{STBY} pin (refer to I_{SB3} in the Electrical Characteristics section of datasheet) it is suggested to add an external capacitance which can filter the eventual current peaks, which could create potential problems of voltage drops in case a very low power external voltage regulator is used. Additional care must be paid on external hardware to limit the current peaks due to the presence of the capacitance (when $\overline{EA}$ functionality is used and the external bipolar is turned on, see *Figure 74*).

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 75*.

All three timers of block GPT1 (T2, T3, T4) can run in three basic modes: Timer, gated timer, and counter mode, and all timers can count either up or down. Each timer has an associated alternate input function pin on Port3, which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) can be programmed by software or can be dynamically altered by a signal at an external control-input pin. Each overflow/underflow of core timer T3 can be indicated on an alternate output function pin. The auxiliary timers T2 and T4 can, additionally, be concatenated with the core timer, or used as capture or reload registers for the core timer.

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

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

Figure 75. SFRs and port pins associated with timer block GPT1

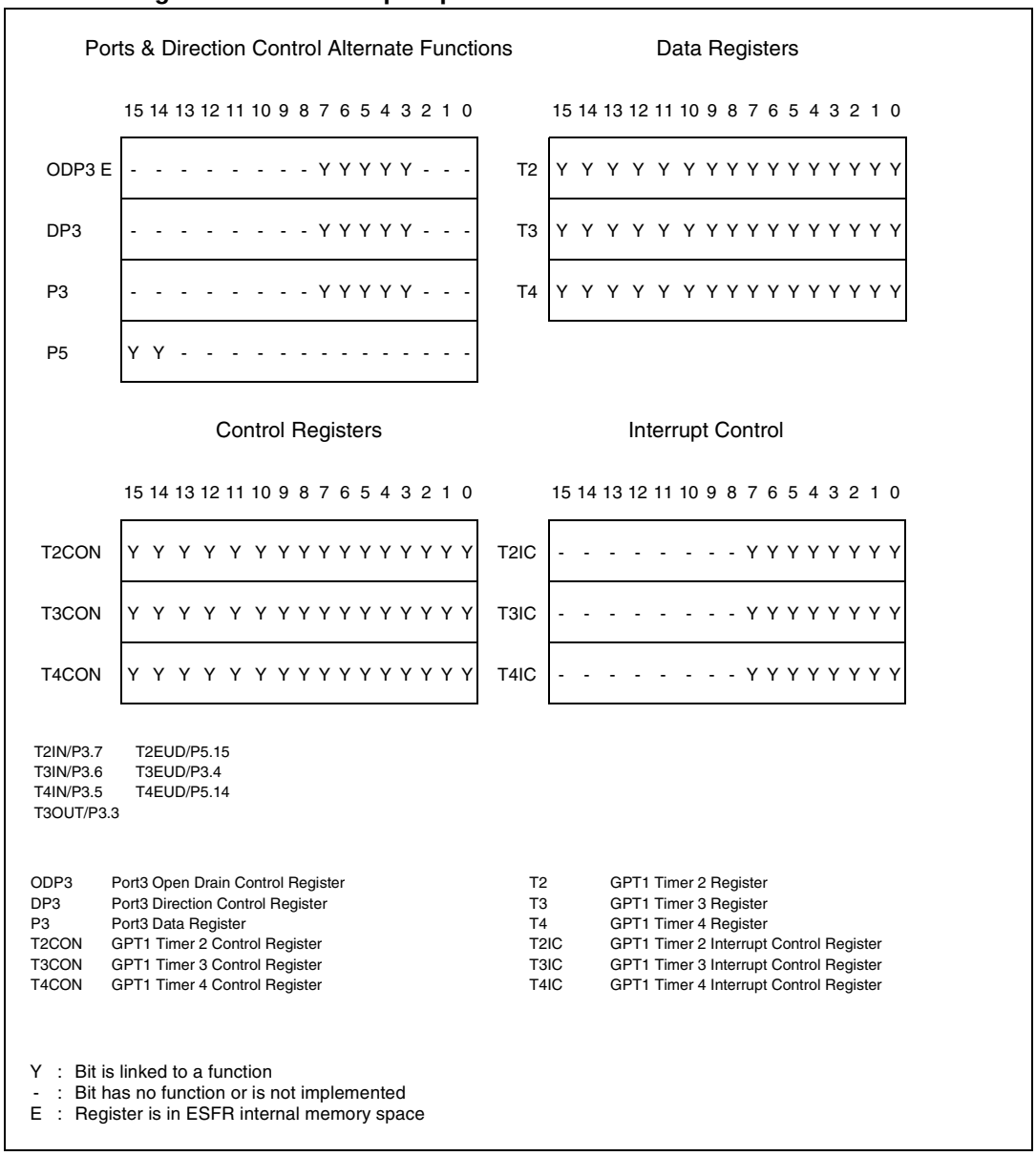
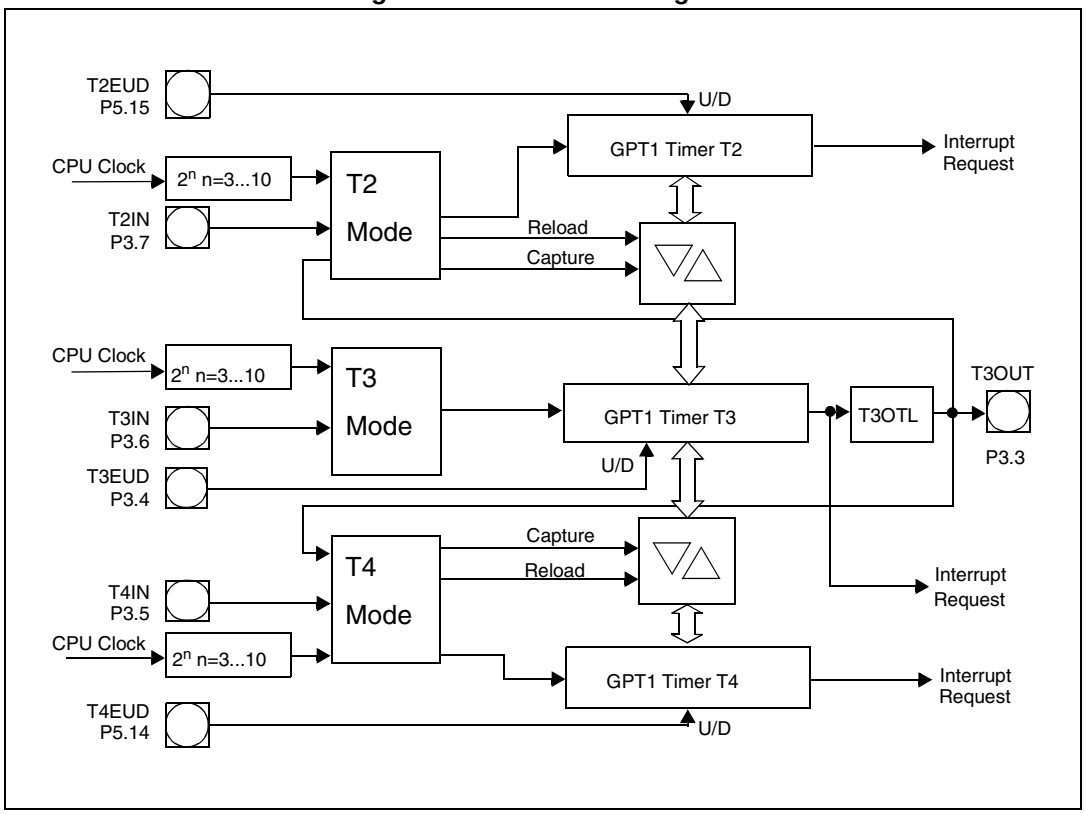


Figure 76. GPT1 block diagram



9.1.1 GPT1 core timer T3

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

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	**T3OTL**	T3OE	T3UDE	T3UD	T3R	T3M			T3I		
					RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
T3I	**Timer 3 Input Selection** - Depends on the operating mode, see respective sections.
T3M	**Timer 3 Mode Control** (Basic Operating Mode) 0 0 0: Timer Mode 0 0 1: Counter Mode 0 1 0: Gated Timer with Gate active low 0 1 1: Gated Timer with Gate active high 1 0 X: Reserved (do not use this combination) 1 1 0: Incremental interface mode 1 1 1: Reserved (do not use this combination)
T3R	**Timer 3 Run bit**: T3R = '0': Timer / Counter 3 stops - T3R = '1': Timer / Counter 3 runs
T3UD	**Timer 3 Up / Down Control** (1)

Bit	Function
T3UDE	**Timer 3 External Up/Down Enable** (1)
T3OE	**Alternate Output Function Enable** T3OE = '0': Alternate Output Function Disabled - T3OE = '1': Alternate Output Function Enabled
T3OTL	**Timer 3 Output Toggle Latch** - Toggles on each overflow / underflow of T3. Can be set or reset by software.

1. For the effects of bit T3UD and T3UDE refer to *Table 33: GPT1 core timer T3 count direction control*.

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 *Table 33*.

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 33. GPT1 core timer T3 count direction control

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

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

Timer 3 output toggle latch

An overflow or underflow of timer T3 will clock the toggle bit T3OTL in control register T3CON. T3OTL can also be set or reset by software.

Bit T3OE (Alternate Output Function Enable) in register T3CON enables the state of T3OTL to be an alternate function of the external output pin T3OUT/P3.3. For that purpose, a '1' must be written into port data latch P3.3 and pin T3OUT/P3.3 must be configured as output by setting direction control bit DP3.3 to '1'. If T3OE = '1', pin T3OUT then outputs the state of T3OTL. If T3OE = '0', pin T3OUT can be used as general purpose I/O pin.

In addition, T3OTL can be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4.

For this purpose, the state of T3OTL does not have to be available at pin T3OUT, because an internal connection is provided for this option.

Timer 3 in timer mode

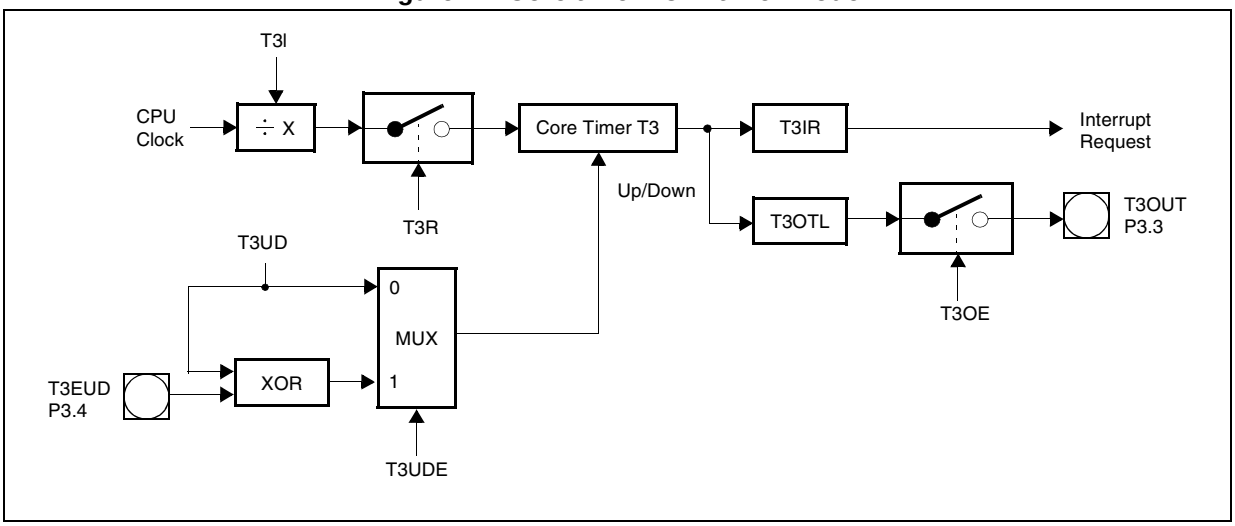
Timer mode for the core timer T3 is selected by setting bit-field T3M in register T3CON to '000b'. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable pre-scaler, which is selected by bit-field T3I.

The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:

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

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

Figure 77. Core timer T3 in timer mode



The timer resolutions which result from the selected pre-scaler option are listed in the *Table 34*. 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 34. 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 78*). 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 35*).

Figure 78. Core timer T3 in gated timer mode

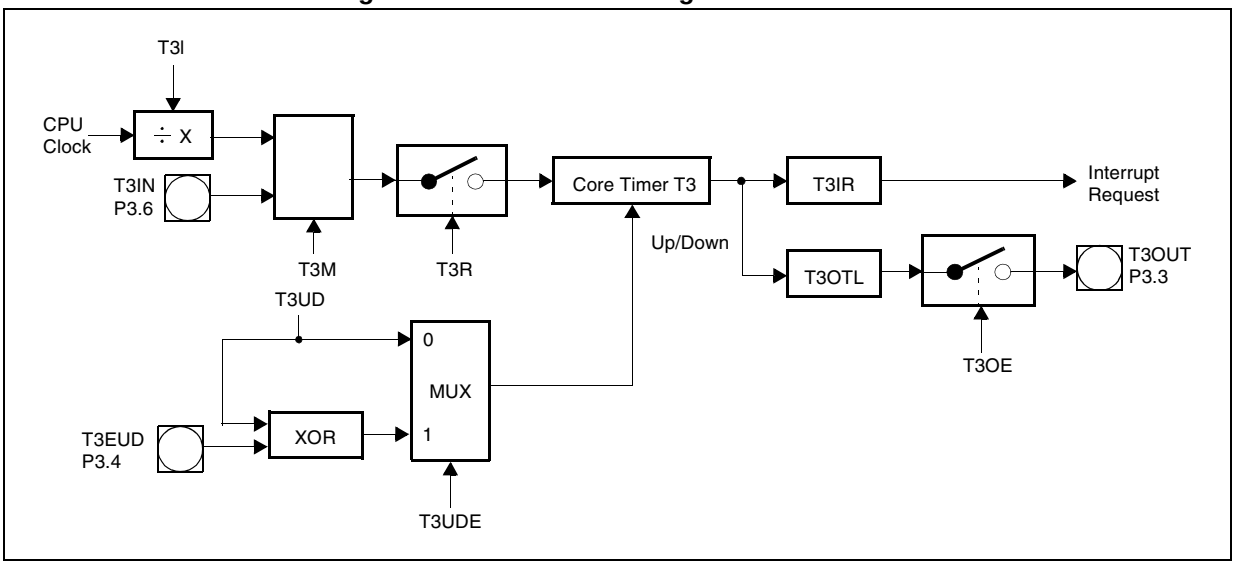


Figure 79. Core timer T3 in counter mode

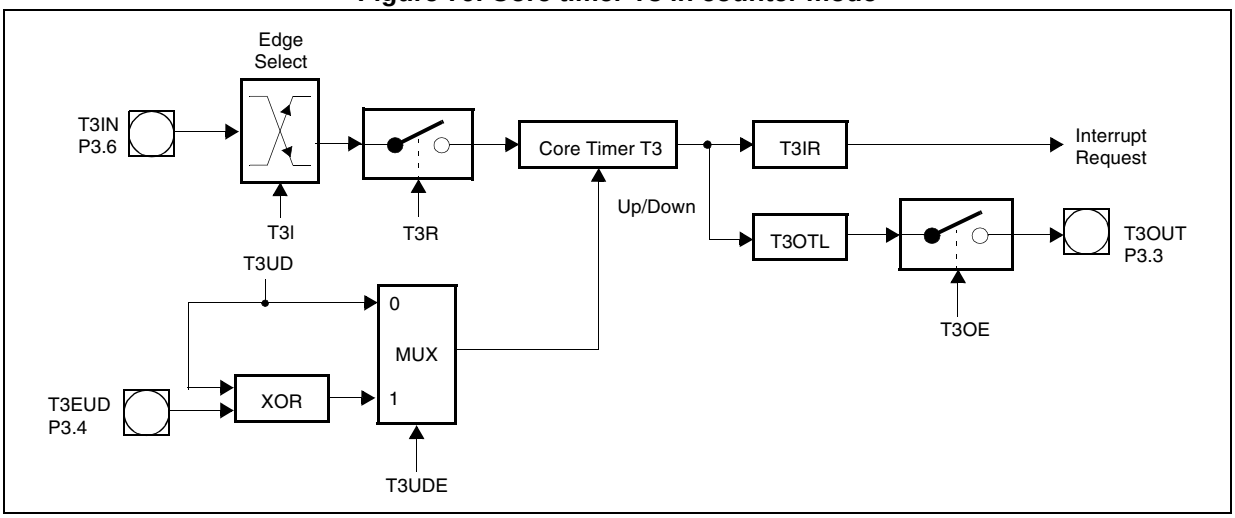


Table 35. GPT1 core timer T3 (counter mode) input edge selection

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, allowed in counter mode, is f_{CPU} / 16.

To ensure that a transition of the count input signal which is applied to T3IN is correctly recognized, its level should be held high or low for at least 8 CPU clock cycles before it changes.

Timer 3 in incremental interface mode

Incremental interface mode for the core timer T3 is selected by setting bit-field T3M in register T3CON to '110b'. In incremental interface mode the two inputs associated with timer T3 (T3IN T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input pins which gives 2-fold or 4-fold resolution to the encoder input (see *Figure 4 on page 41*).

Bit-field T3I in control register T3CON selects the triggering transitions (see *Table 36*). In this mode the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal.

So T3 is modified automatically according to the speed and the direction of the incremental encoder and its contents, therefore, always represent the encoder's current position.

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

The third encoder output 'T0 $\overline{T0}$' which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset timer T3 (for example, via PEC transfer from ZEROS) (see *Figure 81*).

Figure 80. Core timer T3 in incremental interface mode

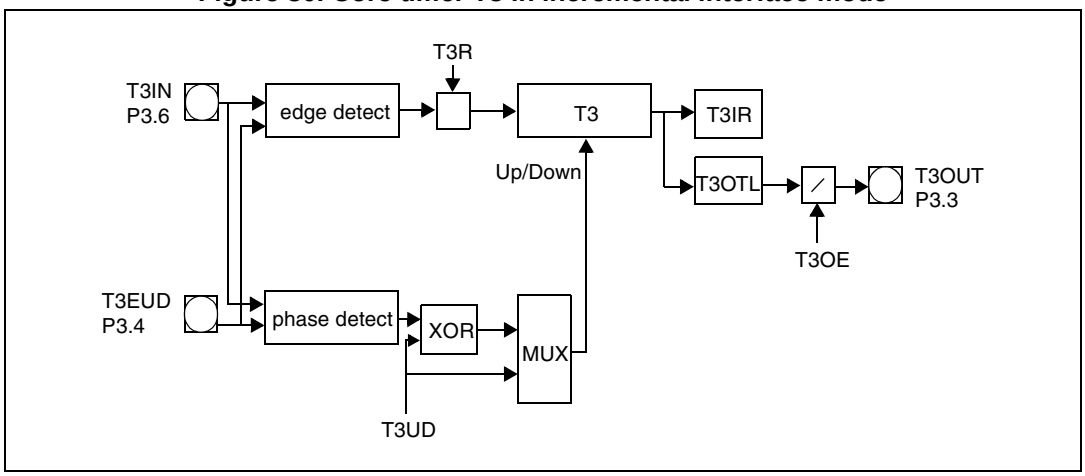
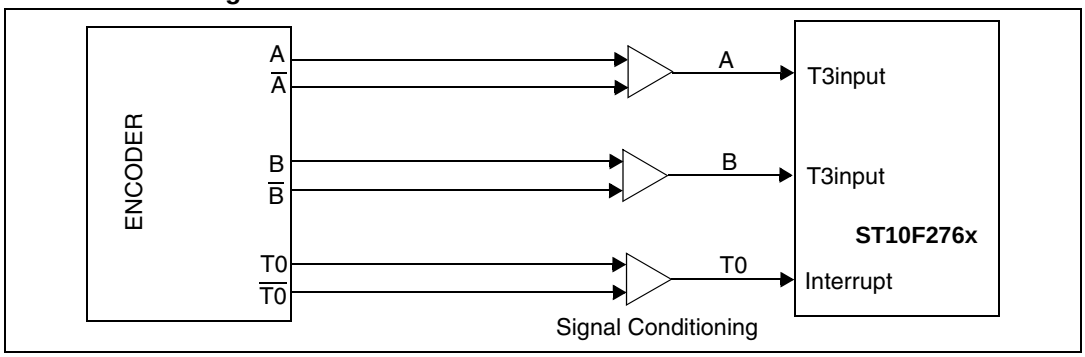


Table 36. GPT1 core timer T3 (incremental interface mode) input edge selection

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 81. Connection of the encoder to the ST10F276



For incremental interface operation the following conditions must be met

- Bit-field T3M must be '110b'
- Both pins T3IN and T3EUD must be configured as input, at the respective direction control bit with '0'.
- Bit T3EUD must be '1' to enable automatic direction control.

The maximum allowed input frequency in incremental interface mode is f_{CPU} / 16. To ensure correct recognition of the transition of any input signal, its level should be held high or low for at least eight CPU clock cycles.

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

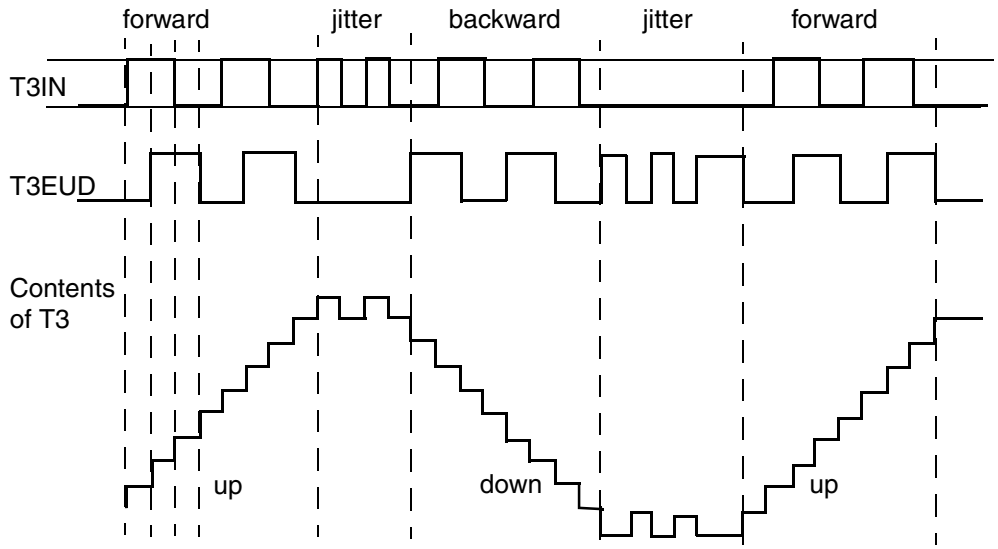
This corresponds to the rotation direction of the connected sensor. The table below summarizes the possible combinations.

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 82* gives examples of T3's operation, visualizing count signal generation and direction control.

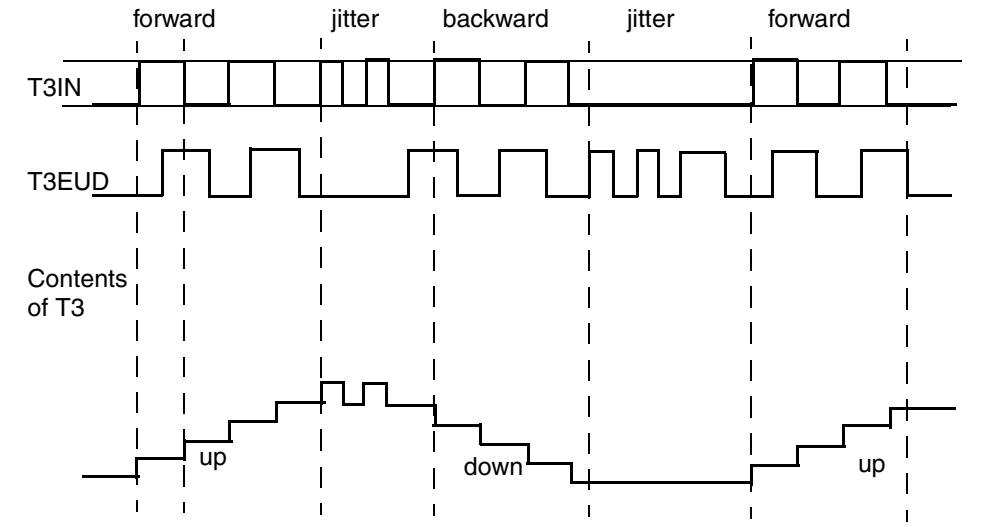
It also shows how input jitter is compensated. This might occur if the sensor stays near to one of the switching points.

Figure 82. Evaluation of the incremental encoder signals



Note: This example shows the timer behavior assuming that T3 counts upon any transition on any input, T3I='011b'

Figure 83. Evaluation of the incremental encoder signals



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

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

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 bit-addressable control registers T2CON and T4CON, which are both organized identically.

Note that functions which are present in all three timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

T2CON (FF40h / A0h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T2UDE	T2UD	T2R	T2M				T2I	
							RW	RW	RW	RW				RW	

T4CON (FF44h / A2h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T4UDE	T4UD	T4R	T4M				T4I	
							RW	RW	RW	RW				RW	

Bit	Function
TxI	**Timer x Input Selection** Depends on the Operating Mode, see respective sections.
TxM	**Timer x Mode Control (Basic Operating Mode)** 0 0 0: Timer Mode 0 0 1: Counter Mode 0 1 0: Gated Timer with Gate active low 0 1 1: Gated Timer with Gate active high 1 0 0: Reload Mode 1 0 1: Capture Mode 1 1 0: Incremental interface mode 1 1 1: Reserved (do not use this combination)
TxR	**Timer x Run bit** TxR = '0': Timer / Counter x stops TxR = '1': Timer / Counter x runs
TxUD	**Timer x Up / Down Control** (1)
TxUDE	**Timer x External Up/Down Enable** (1)

1. For the effects of bit TxUD and TxUDE refer to the direction Table 35 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 84*).

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 37*).

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 eight CPU clock cycles before it changes.

Figure 84. Auxiliary timer in counter mode

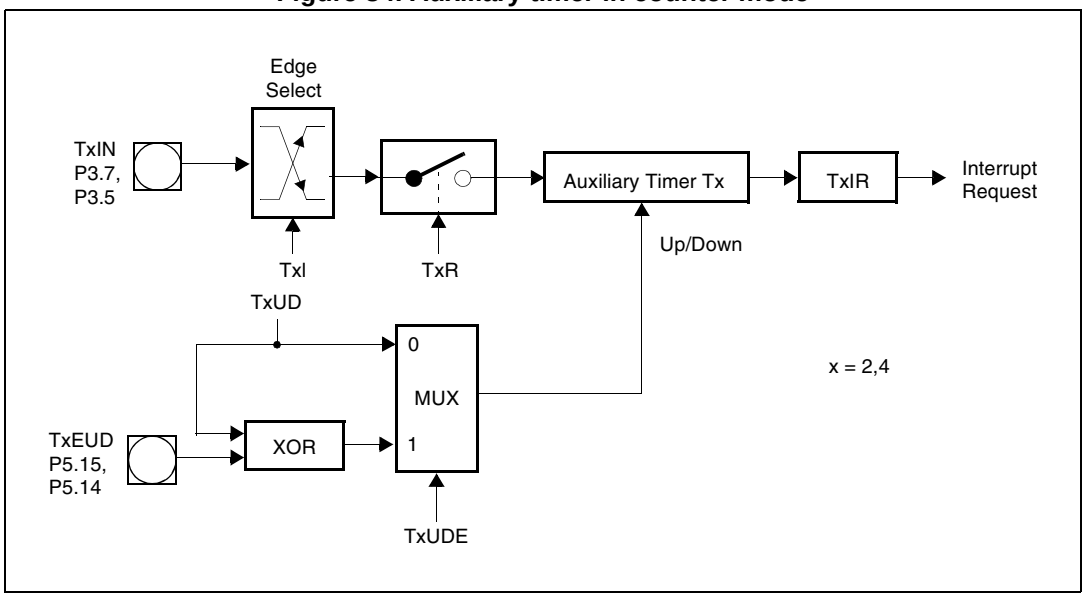


Table 37. GPT1 auxiliary timer (counter mode) input edge selection

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

Table 37. GPT1 auxiliary timer (counter mode) input edge selection

T2I / T4I	Triggering edge for counter increment / decrement
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.

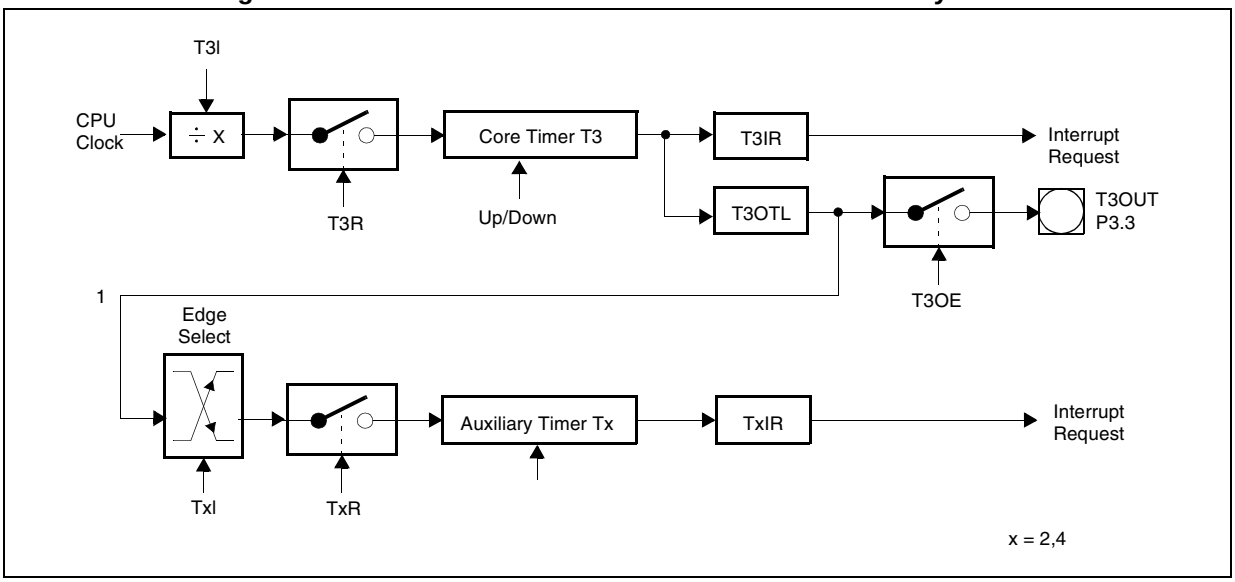
T3 can operate in timer, gated timer or counter mode in this case (see *Figure 85*).

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 37*). A transition of the auxiliary timer’s input or the output toggle latch T3OTL may trigger the reload.

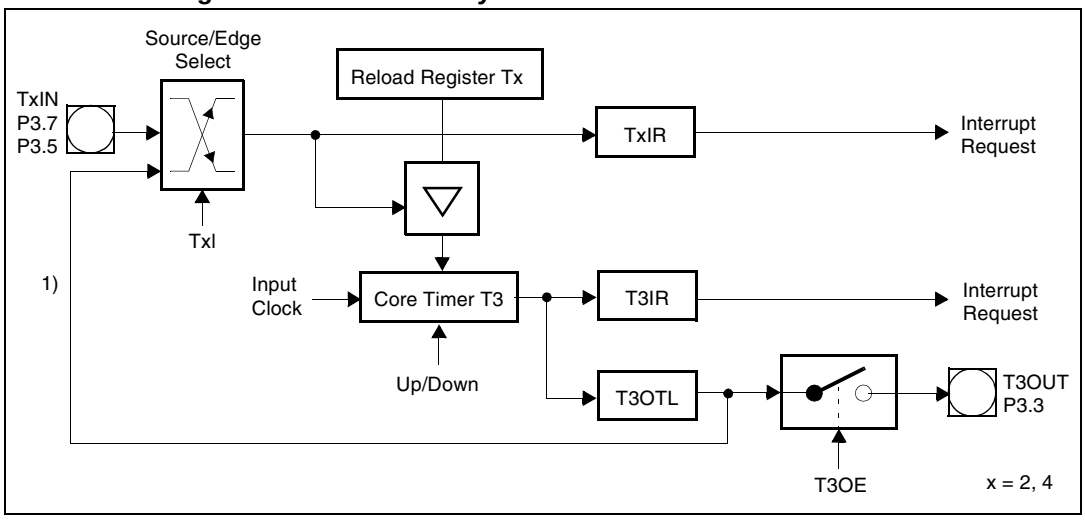
Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.

Figure 85. 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 86. GPT1 auxiliary timer in reload mode



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

Upon a trigger signal T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

Note: When a T3OTL transition is selected for the trigger signal, also the interrupt request flag T3IR will be set upon a trigger, indicating T3's overflow or underflow.
Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

The reload mode triggered by T3OTL can be used in a number of different configurations. Depending on the selected active transition the following functions can be performed:

- If both a positive and a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this “single-transition” mode for both auxiliary timers allows to perform very flexible pulse width modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

Figure 87 shows an example for the generation of a PWM signal using the alternate reload mechanism.

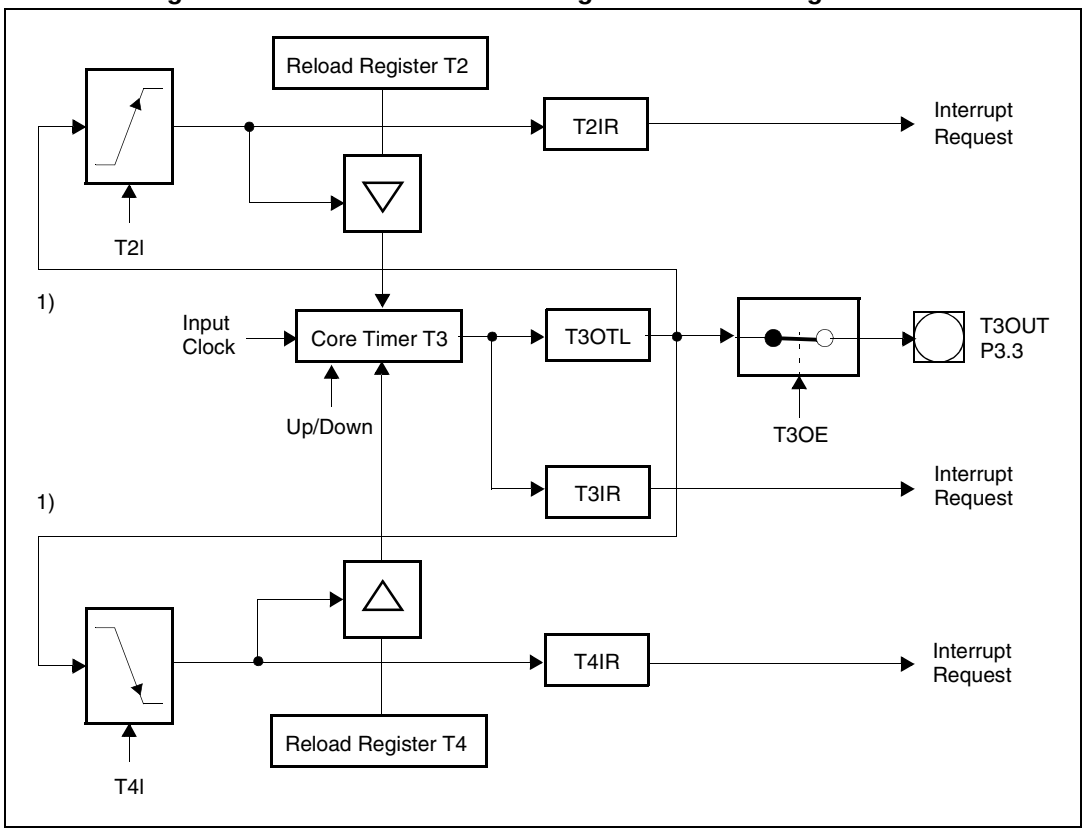
T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions).

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 87. GPT1 timer reload configuration for PWM generation



Note: 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’.

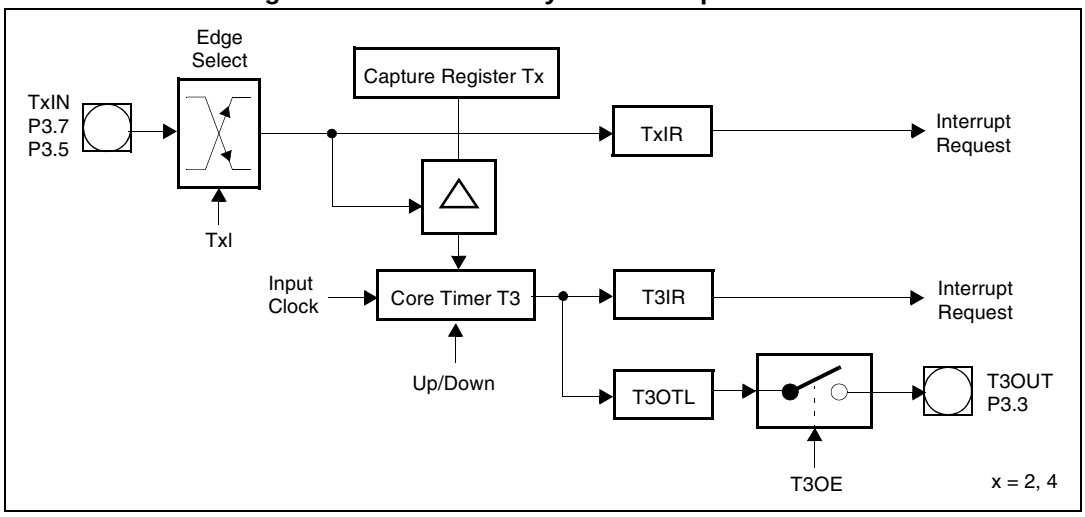
In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN.

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 88. GPT1 auxiliary timer in capture mode



LUpon 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 eight 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: Details of the control fields is given in the general Interrupt Control Register description.

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 1 on page 23*.

Timer block GPT2 supports high precision event control with a maximum resolution of four CPU clock cycles. It includes the two timers T5 and T6, and the 16-bit capture/reload register CAPREL. Timer T6 is referred to as the core timer, and T5 is referred to as the auxiliary timer of GPT2.

Each timer has an alternate associated input pin which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) may be programmed via software or may be dynamically altered by a signal at an external control input pin. An overflow/underflow of T6 is indicated by the output toggle bit T6OTL whose state may be output on an alternate function port pin. In addition, T6 may be reloaded with the contents of CAPREL.

The toggle bit also supports the concatenation of T6 with auxiliary timer T5, while concatenation of T6 with the timers of the CAPCOM units is provided through a direct connection.

Triggered by an external signal, the contents of T5 can be captured into register CAPREL, and T5 may optionally be cleared. Both timer T6 and T5 can count up or down, and the current timer value can be read or modified by the CPU in the non bit-addressable SFRs T5 and T6.

Figure 89. SFRs and port pins associated with timer block GPT2

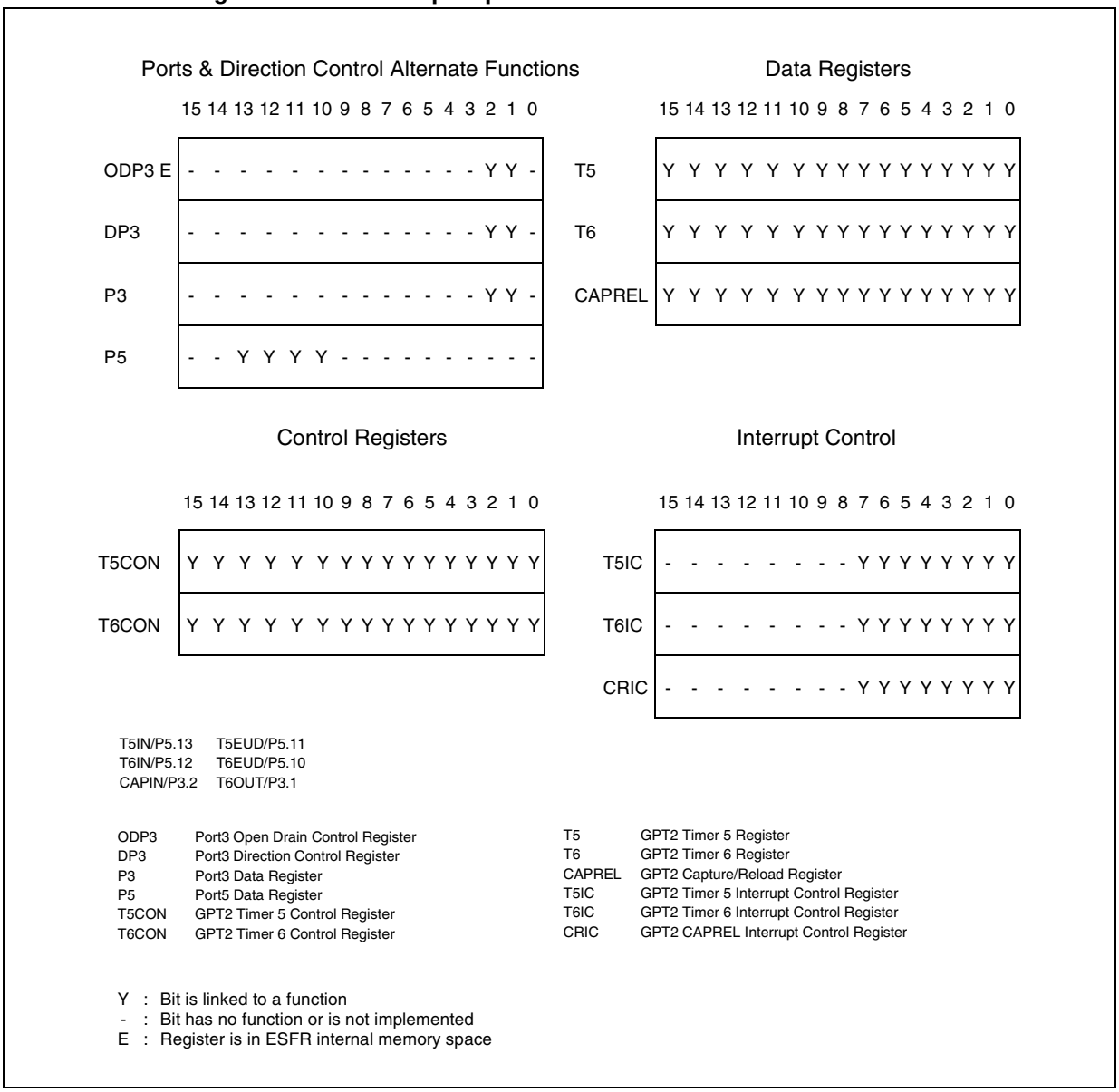
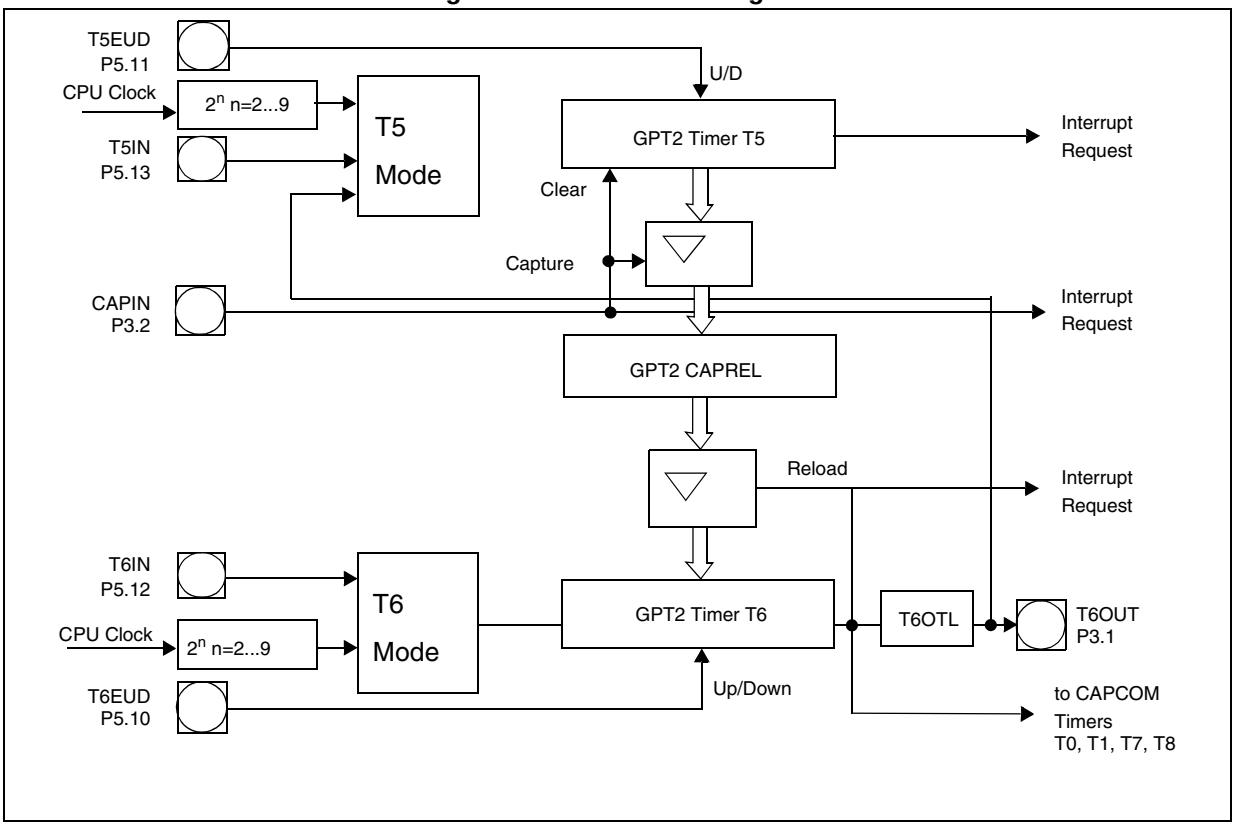


Figure 90. GPT2 block diagram



9.2.1 GPT2 core timer T6

The operation of the core timer T6 is controlled by its bit-addressable control register T6CON.

T6CON (FF48h / A4h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T6SR	-	-	-	-	**T6OTL**	T6OE	T6UDE	T6UD	T6R	T6M				T6I	
RW					RW	RW	RW	RW	RW	RW				RW	

Bit	Function
T6I	**Timer 6 Input Selection** Depends on the Operating Mode, see respective sections.
T6M	**Timer 6 Mode Control** (Basic Operating Mode) 0 0 0: Timer Mode 0 0 1: Counter Mode 0 1 0: Gated Timer with Gate active low 0 1 1: Gated Timer with Gate active high 1 X X: Reserved. Do not use this combination.
T6R	**Timer 6 Run bit** T6R = '0':Timer / Counter 6 stops T6R = '1':Timer / Counter 6 runs
T6UD	**Timer 6 Up / Down Control** (1)

Bit	Function
T6UDE	**Timer 6 External Up/Down Enable** (1)
T6OE	**Alternate Output Function Enable** T6OE = '0': Alternate Output Function Disabled T6OE = '1': Alternate Output Function Enabled
T6OTL	**Timer 6 Output Toggle Latch** Toggles on each overflow / underflow of T6. Can be set or reset by software.
T6SR	**Timer 6 Reload Mode Enable** T6SR = '0': Reload from register CAPREL Disabled T6SR = '1': Reload from register CAPREL Enabled

1. *For the effects of bit T6UD and T6UDE refer to the direction Table 38.*

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 38*. 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 38. 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.

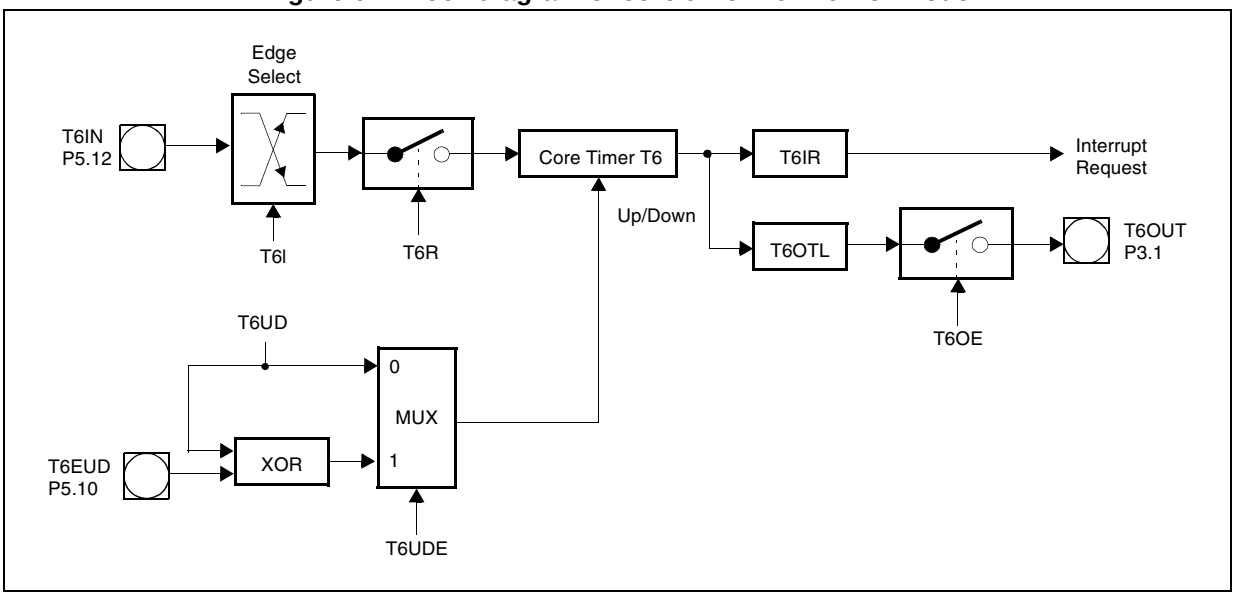
An overflow or underflow of timer T6 can also be used to clock the timers in the CAPCOM units. For this purpose, there is a direct internal connection between timer T6 and the CAPCOM timers.

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:

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

Figure 91. 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 39*. This table also applies to the Gated Timer Mode of T6 and to the auxiliary timer T5 in timer and gated timer mode.

Table 39. 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 92*). 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 40*).

Figure 92. Block diagram of core timer T6 in gated timer mode

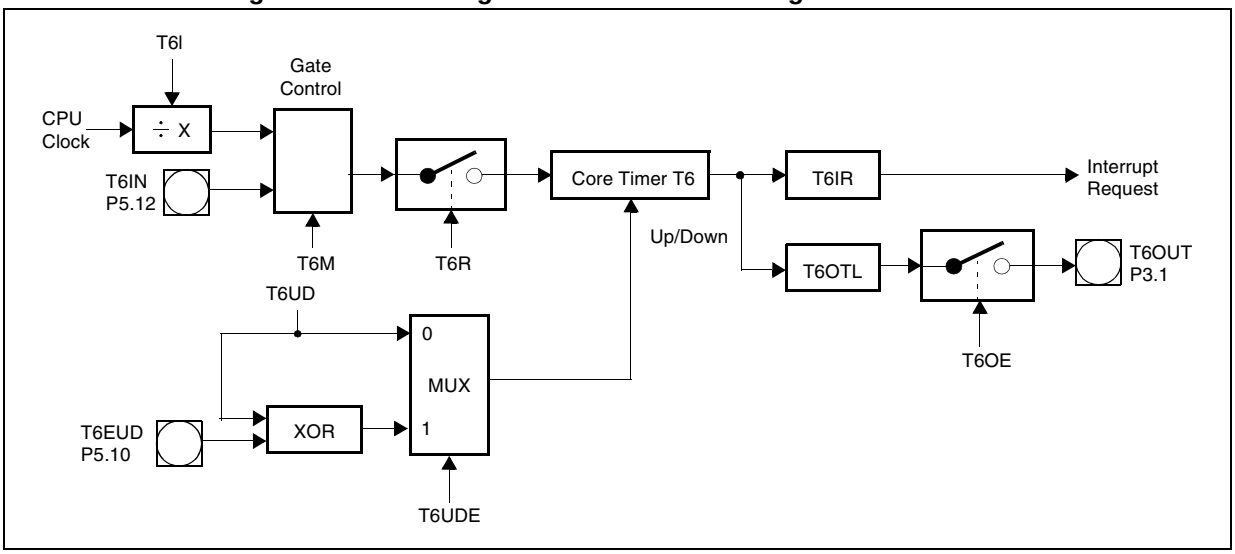


Figure 93. Block diagram of core timer T6 in counter mode

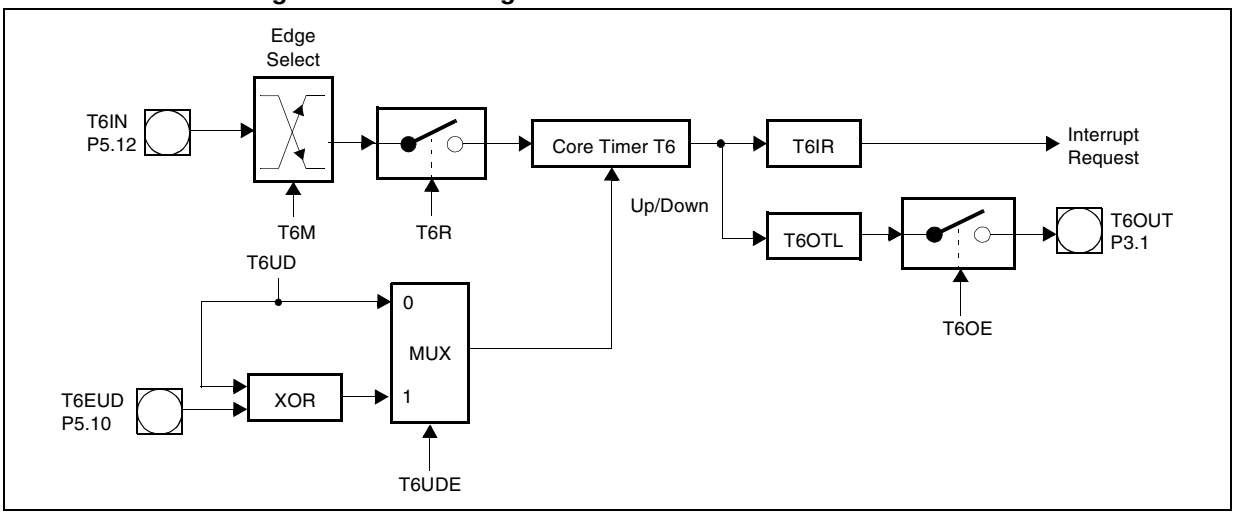


Table 40. GPT2 core timer T6 (counter mode) input edge selection

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 four 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 three counting modes, the auxiliary timer can be concatenated with the core timer.

The auxiliary timer has no output toggle latch and no alternate output function. The individual configuration for timer T5 is determined by its bit-addressable control register T5CON. Note that functions which are present in both timers of block GPT2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

T5CON (FF46h / A3h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T5SC	T5CLR	CI		-	-	CT3	T5UDE	T5UD	T5R	-	T5M			T5I	
RW	RW	RW				RW	RW	RW	RW		RW			RW	

Bit	Function
T5I	**Timer 5 Input Selection** Depends on the Operating Mode, see respective sections.
T5M	**Timer 5 Mode Control (Basic Operating Mode)** 0 0: Timer Mode 0 1: Counter Mode 1 0: Gated Timer with Gate active low 1 1: Gated Timer with Gate active high
T5R	**Timer 5 Run bit** T5R = '0': Timer / Counter 5 stops T5R = '1': Timer / Counter 5 runs
T5UD	Timer 5 Up / Down Control (1)
T5UDE	Timer 5 External Up/Down Enable (1)
CT3	Capture Trigger 3 0: Inactive 1: Capture on Timer3 events
CI	**Register CAPREL Input Selection** 0 0: Capture disabled 0 1: Positive transition (rising edge) on CAPIN 1 0: Negative transition (falling edge) on CAPIN 1 1: Any transition (rising or falling edge) on CAPIN
T5CLR	**Timer 5 Clear bit** T5CLR = '0': Timer 5 not cleared on a capture T5CLR = '1': Timer 5 is cleared on a capture
T5SC	**Timer 5 Capture Mode Enable** T5SC = '0': Capture into register CAPREL Disabled T5SC = '1': Capture into register CAPREL Enabled

1. *For the effects of bit TxUD and TxUDE refer to the direction Table 38.*

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 94*). Bit-field T5I in control register T5CON selects the triggering transition (see *Table 41*).

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 four CPU clock cycles before it changes.

Figure 94. Block diagram of auxiliary timer T5 in counter mode

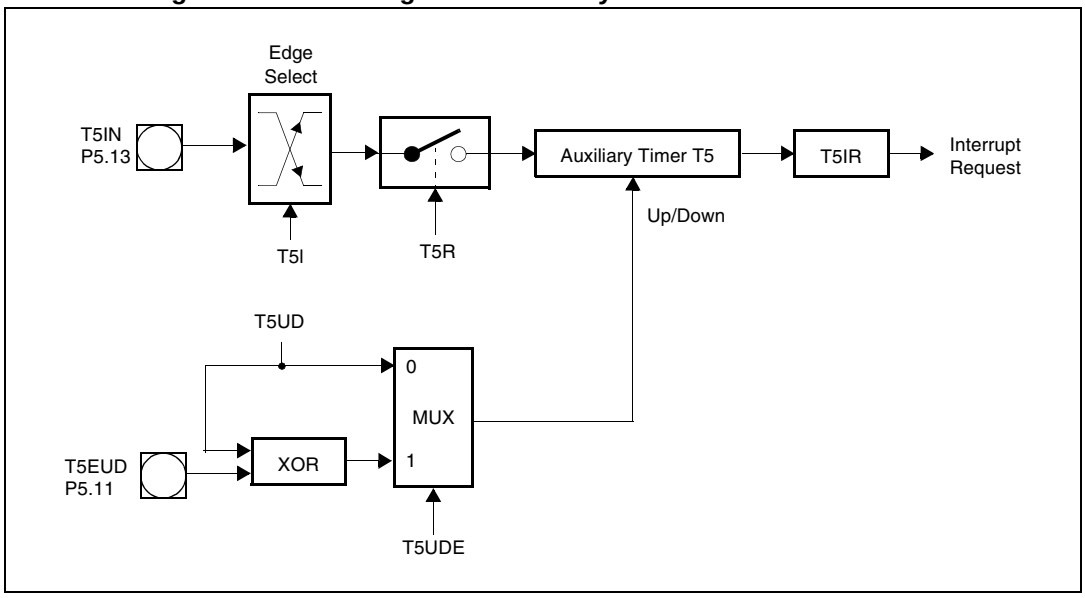


Table 41. GPT2 auxiliary timer (counter mode) input edge selection

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 95*).

GPT2 capture / reload register CAPREL in capture mode

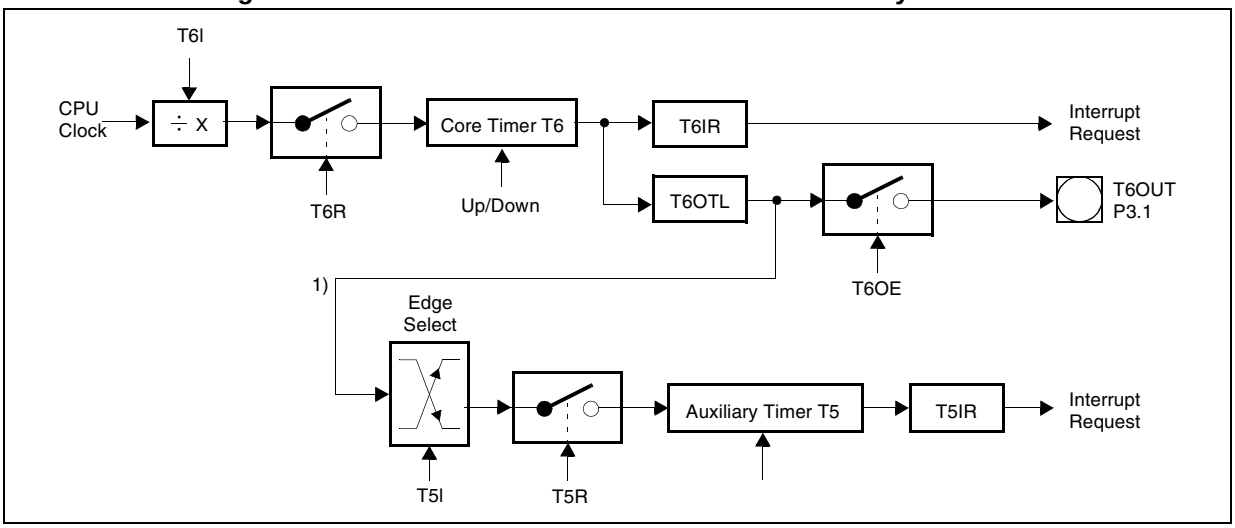
This 16-bit register can be used as a capture register for the auxiliary timer T5. This mode is selected by setting bit T5SC = '1' in control register T5CON. Bit CT3 selects the external input pin CAPIN or the input pins of timer T3 as the source for a capture trigger. Either a positive, a negative, 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 four 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.

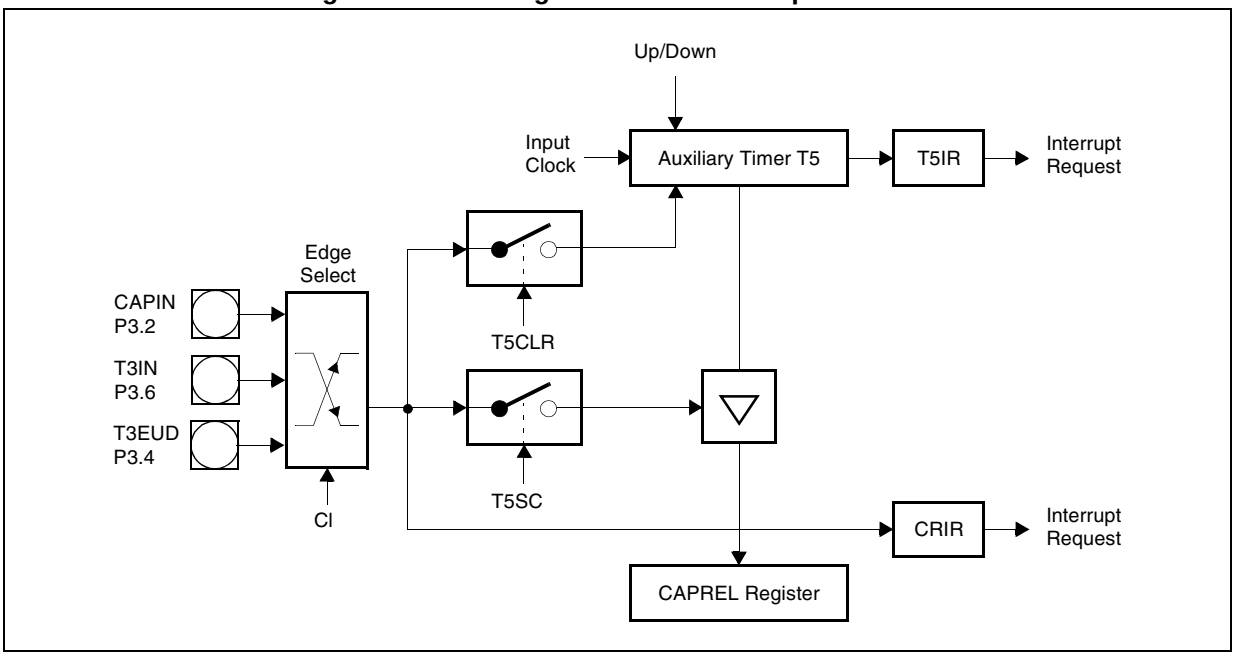
Note: Bit T5SC only controls whether a capture is performed or not. If T5SC = '0', the input pin CAPIN can still be used to clear timer T5 or as an external interrupt input. This interrupt is controlled by the CAPREL interrupt control register CRIC (see Figure 96).

Figure 95. Concatenation of core timer T6 and auxiliary timer T5



Note: Line only affected by over/underflows of T6, but NOT by software modifications of T6OTL.

Figure 96. GPT2 register CAPREL in capture mode

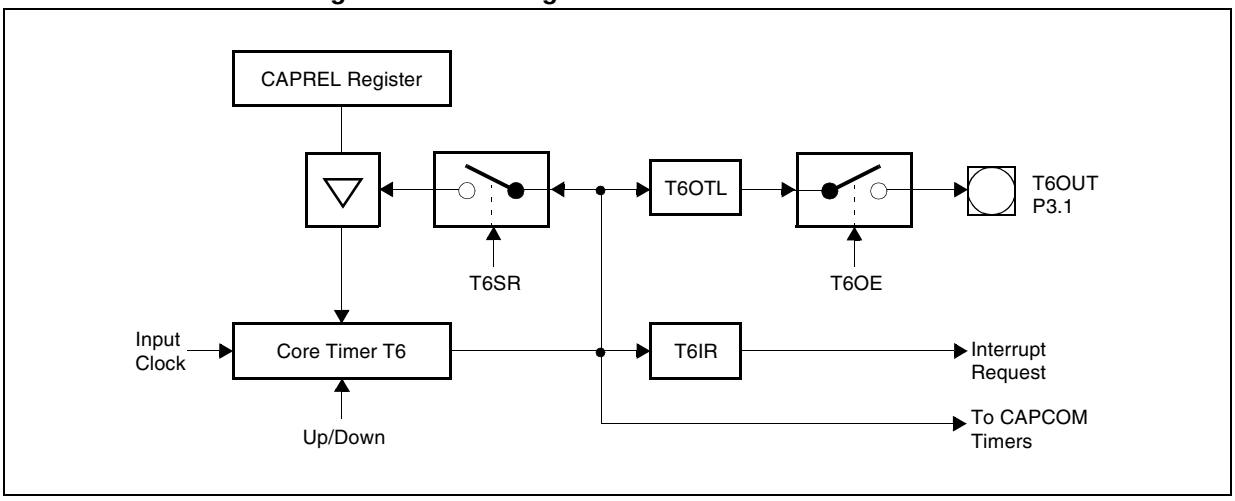


GPT2 capture / reload register CAPREL in reload mode

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 97. GPT2 register CAPREL in reload mode



GPT2 capture / reload register CAPREL in capture-and-reload mode

Since the reload function and the capture function of register CAPREL can be enabled individually by bit T5SC and T6SR, the two functions can be enabled simultaneously by setting both bits. This feature can be used to generate an output frequency that is a multiple of the input frequency (see *Figure 98*).

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.

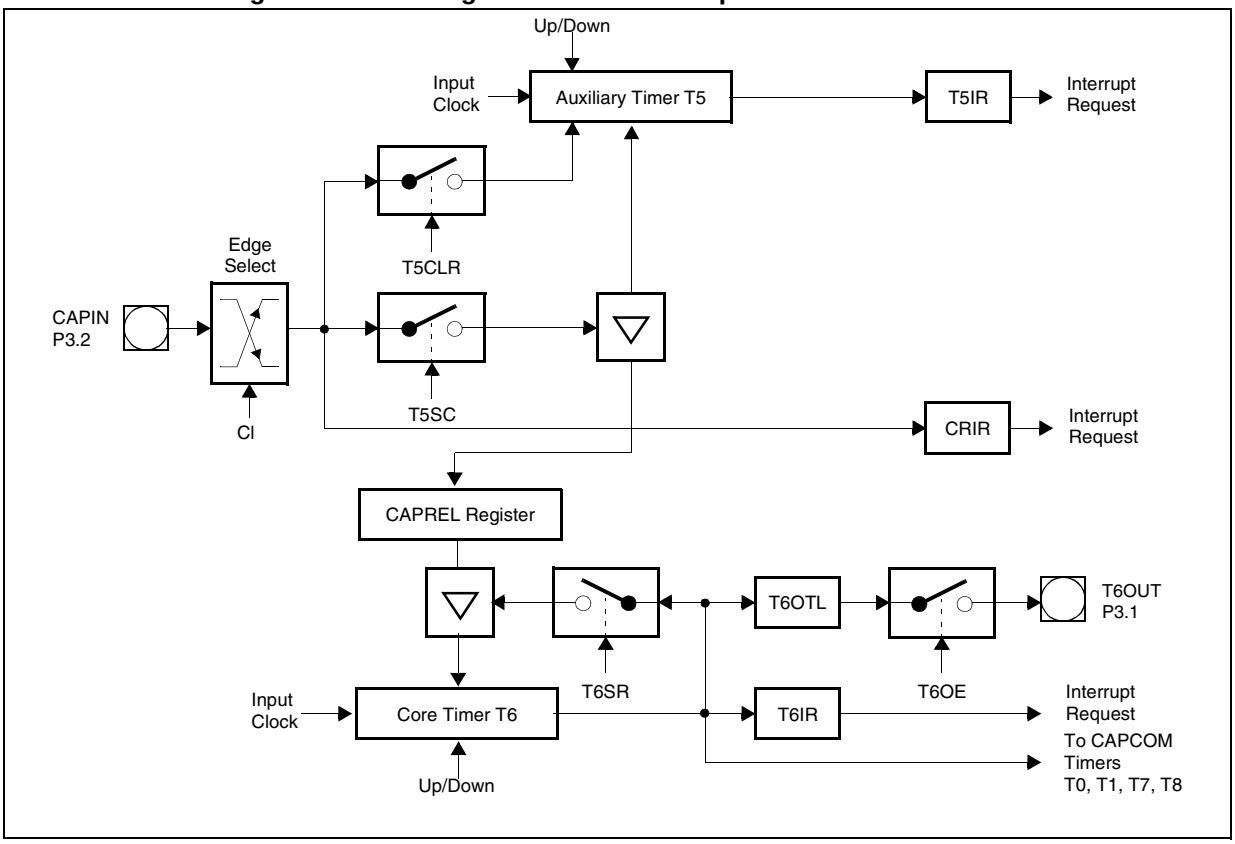
For example, Timer T5 runs in timer mode counting up with a frequency of f_{CPU} / 32. The external events are applied to pin CAPIN. When an external event occurs, the timer T5 contents are latched into register CAPREL, and timer T5 is cleared (T5CLR = ‘1’).

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 eight times faster than timer T5, it will underflow eight times within the time between two external events.

Thus, the underflow signal of timer T6 generates eight '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 eight 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 98. 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 (counting up), or when it underflows from 0000h to FFFFh (counting down), its interrupt request flag (T5IR or T6IR) in register TxIC will be set. Whenever a transition according to the selection in bit-field CI is detected at pin CAPIN, interrupt request flag CRIR in register CRIC is set. Setting any request flag will cause an interrupt to the respective timer or CAPREL interrupt vector (T5INT, T6INT or CRINT) or trigger a PEC service, if the respective interrupt enable bit (T5IE or T6IE in register TxIC, CRIE in register CRIC) is set. There is an interrupt control register for each of the two timers and for the CAPREL register.

T5IC (FF66h / B3h) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T5IR	T5IE	ILVL				GLVL	
								RW	RW	RW				RW	

T6IC (FF68h / B4h) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T6IR	T6IE	ILVL				GLVL	
								RW	RW	RW				RW	

CRIC (FF6Ah / B5h) SFR Reset Value: - - 00h

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

Note: Refer to Chapter 5.1.3 for explanation of the control fields.

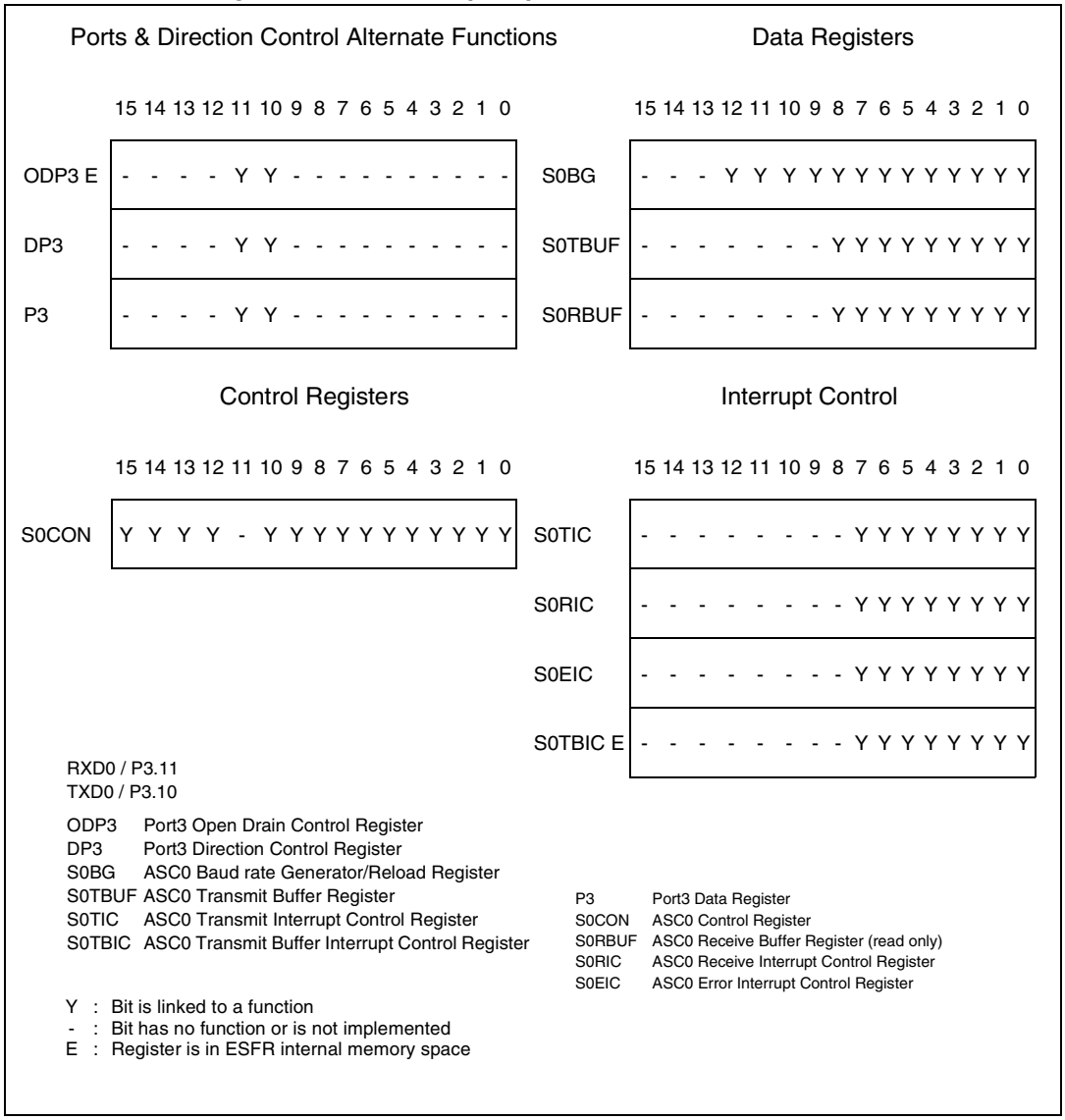
10 Asynchronous / synchronous serial interface

The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between the ST10F276 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 ST10F276. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered.

For multiprocessor communication, a mechanism to distinguish address from data byte is included. Testing is supported by a loop-back option. A 13-bit Baud rate generator provides the ASC0 with a separate serial clock signal.

Figure 99. SFRs and port pins associated with ASC0



The operating mode of the serial channel ASC0 is controlled by its bit-addressable control register S0CON. This register contains control bit for mode and error check selection, and status flags for error identification.

S0CON (FFB0h / D8h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S0R	S0LB	S0BRS	S0ODD	-	**S0OE**	**S0FE**	**S0PE**	S0OEN	S0FEN	S0PEN	S0REN	S0STP	S0M		
RW	RW	RW	RW		RW	RW	RW	RW	RW	RW	RW	RW	RW		

Bit	Function
S0M	**ASC0 Mode Control** 0 0 0: 8-bit data synchronous operation 0 0 1: 8-bit data asynchronous operation 0 1 0: Reserved. Do not use this combination 0 1 1: 7-bit data + parity asynchronous operation 1 0 0: 9-bit data asynchronous operation 1 0 1: 8-bit data + wake up bit asynchronous operation 1 1 0: Reserved. Do not use this combination 1 1 1: 8-bit data + parity asynchronous operation
S0STP	**Number of Stop bit Selection asynchronous operation** 0: One stop bit 1: Two stop bit
S0REN	**Receiver Enable bit** 0: Receiver disabled 1: Receiver enabled (Reset by hardware after reception of byte in synchronous mode)
S0PEN	**Parity Check Enable bit** asynchronous operation 0: Ignore parity 1: Check parity
S0FEN	**Framing Check Enable bit** asynchronous operation 0: Ignore framing errors 1: Check framing errors
S0OEN	**Overrun Check Enable bit** 0: Ignore overrun errors 1: Check overrun errors
S0PE	**Parity Error Flag** Set by hardware on a parity error (S0PEN = ‘1’). Must be reset by software.
S0FE	**Framing Error Flag** Set by hardware on a framing error (S0FEN = ‘1’). Must be reset by software.
S0OE	**Overrun Error Flag** Set by hardware on an overrun error (S0OEN = ‘1’). Must be reset by software.
S0ODD	**Parity Selection bit** 0: Even parity (parity bit set on odd number of ‘1’s in data) 1: Odd parity (parity bit set on even number of ‘1’s in data)
S0BRS	**Baud rate Selection bit** 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3rd

Bit	Function
S0LB	**Loopback Mode Enable bit** 0: Standard transmit/receive mode 1: Loopback mode enabled
S0R	**Baud rate Generator Run bit** 0: Baud rate generator disabled (ASC0 inactive) 1: Baud rate generator enabled

A transmission is started by writing to the Transmit Buffer register S0TBUF (via an instruction or a PEC data transfer).

Only the number of data bits which is determined by the selected operating mode will actually be transmitted. Bits written to positions 9 through 15 of register S0TBUF are always insignificant. After a transmission has been completed, the transmit buffer register is cleared to 0000h.

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

Data reception is enabled by the Receiver Enable bit S0REN. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) Receive Buffer register S0RBUF.

Bits in the upper half of S0RBUF which are not valid in the selected operating mode will be read as zeros.

Data reception is double-buffered, so that reception of a second character may already begin before the previously received character has been read out of the receive buffer register.

In all modes, receive buffer overrun error detection can be selected through bit S0OEN. When enabled, the overrun error status flag S0OE and the error interrupt request flag S0EIR will be set when the receive buffer register has not been read by the time reception of a second character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit S0LB) allows the data currently being transmitted to be received simultaneously in the receive buffer.

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

Note: Serial data transmission or reception is only possible when the Baud rate Generator Run bit S0R is set to '1'. Otherwise the serial interface is idle.
Do not program the mode control field S0M in register S0CON to one of the reserved combinations to avoid unpredictable behavior of the serial interface.

S0TBUF (FEB0h / 58h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	Transmit Data Buffer								
							RW								

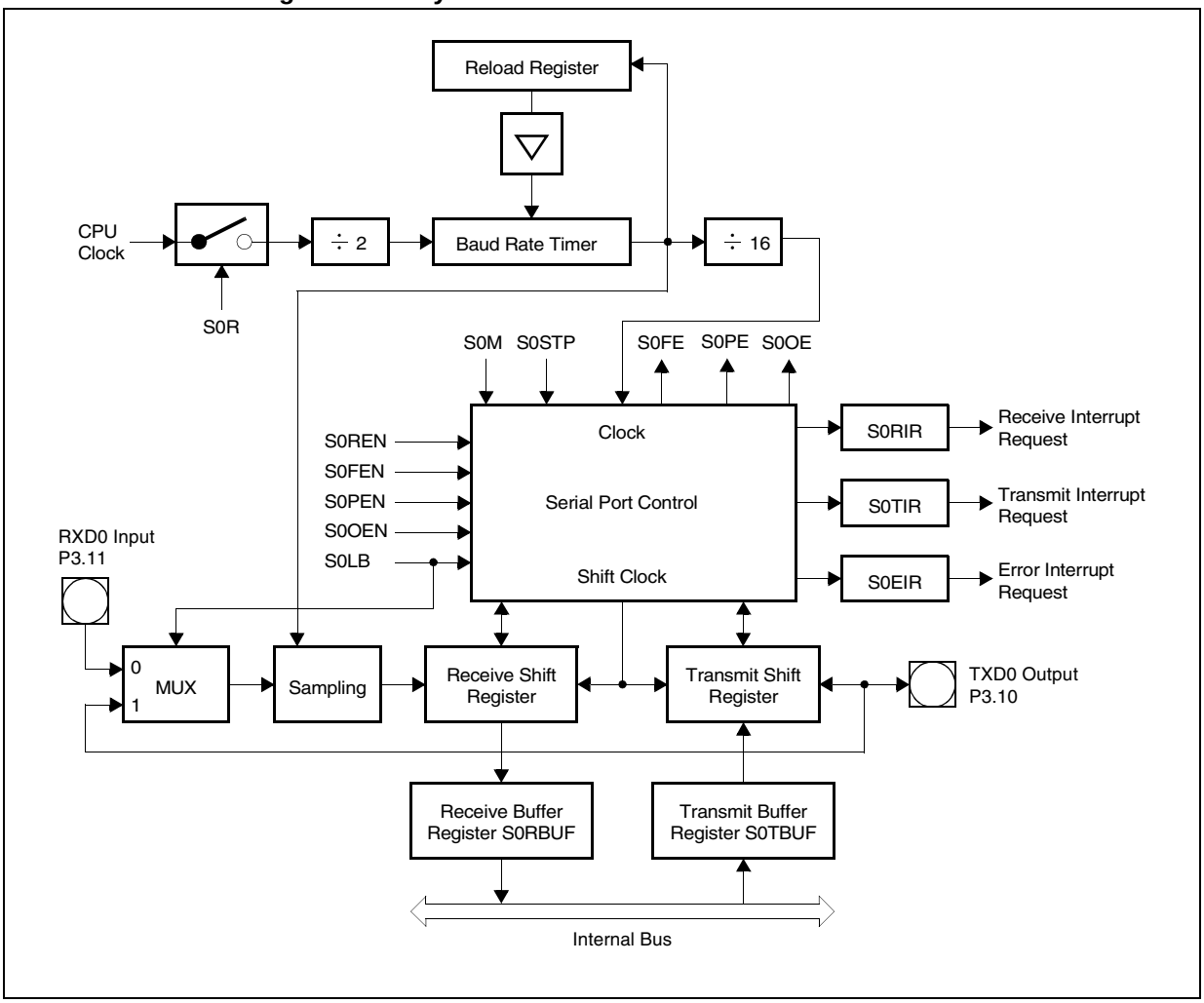
S0RBUF (FEB2h / 59h) SFR Reset Value: 0xxxh

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

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 Port3 pins.

Figure 100. Asynchronous mode of serial channel ASC0

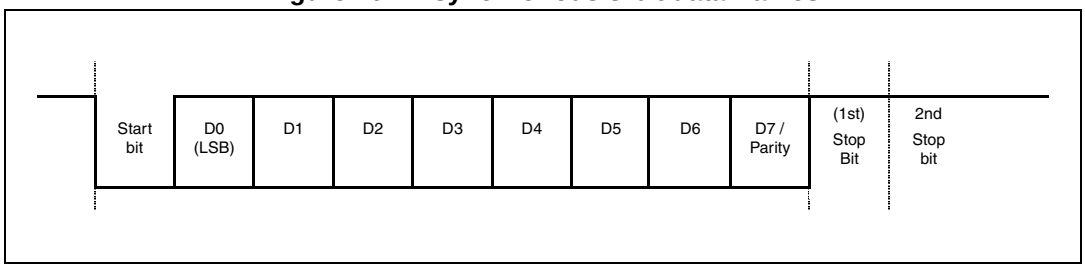


Asynchronous data frames

8-bit data frames either consist of 8 data bits D7...D0 (S0M = ‘001b’), or of 7 data bits D6...D0 plus an automatically generated parity bit (S0M = ‘011b’). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the

modulo-2-sum of the 7 data bits is ‘1’. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 8-bit data mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.

Figure 101. Asynchronous 8-bit data frames



9-bit data frames either consist of 9 data bits D8...D0 (S0M = ‘100b’), of 8 data bits D7...D0 plus an automatically generated parity bit (S0M = ‘111b’) or of 8 data bits D7...D0 plus wake-up bit (S0M = ‘101b’). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is ‘1’. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 9-bit data and wake-up mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit 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 102*).

Asynchronous transmission begins at the next overflow of the divide-by-16 counter (see *Figure 102*), provided that S0R is set and data has been loaded into S0TBUF. The transmitted data frame consists of three basic elements:

- the start bit,
- the data field (8 or 9 bits, LSB first, including a parity bit, if selected),
- the delimiter (1 or 2 stop bits).

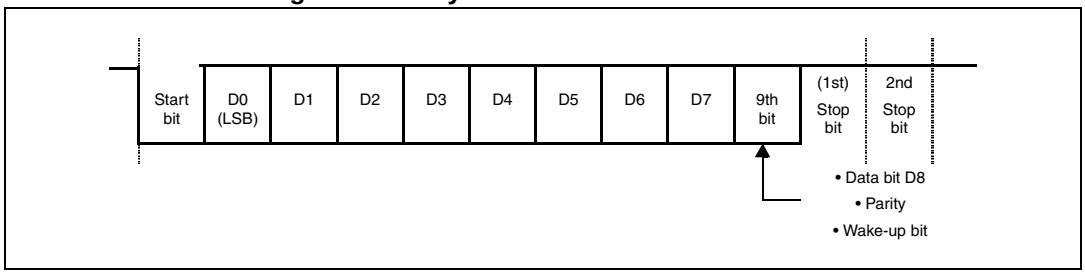
Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag S0TIR will be set before the last bit of a frame is transmitted, that means before the first or the second stop bit is shifted out of the transmit shift register.
The transmitter output pin TXD0/P3.10 must be configured for alternate data output, P3.10 = ‘1’ and DP3.10 = ‘1’.

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bit S0R and S0REN are set. The receive data input pin RXD0 is sampled at 16 times the rate of the selected Baud rate. A majority decision of the 7th, 8th and 9th sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a '0' when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin RXD0. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

Figure 102. Asynchronous 9-bit data frames



When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request flag S0RIR is set after the 9th sample in the last stop bit time slot (as programmed), regardless whether valid stop bit have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input pin.

The receiver input pin RXD0/P3.11 must be configured for input, using direction control register DP3.11 = ‘0’.

Asynchronous reception is stopped by clearing bit S0REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bit that follow this frame will not be recognized.

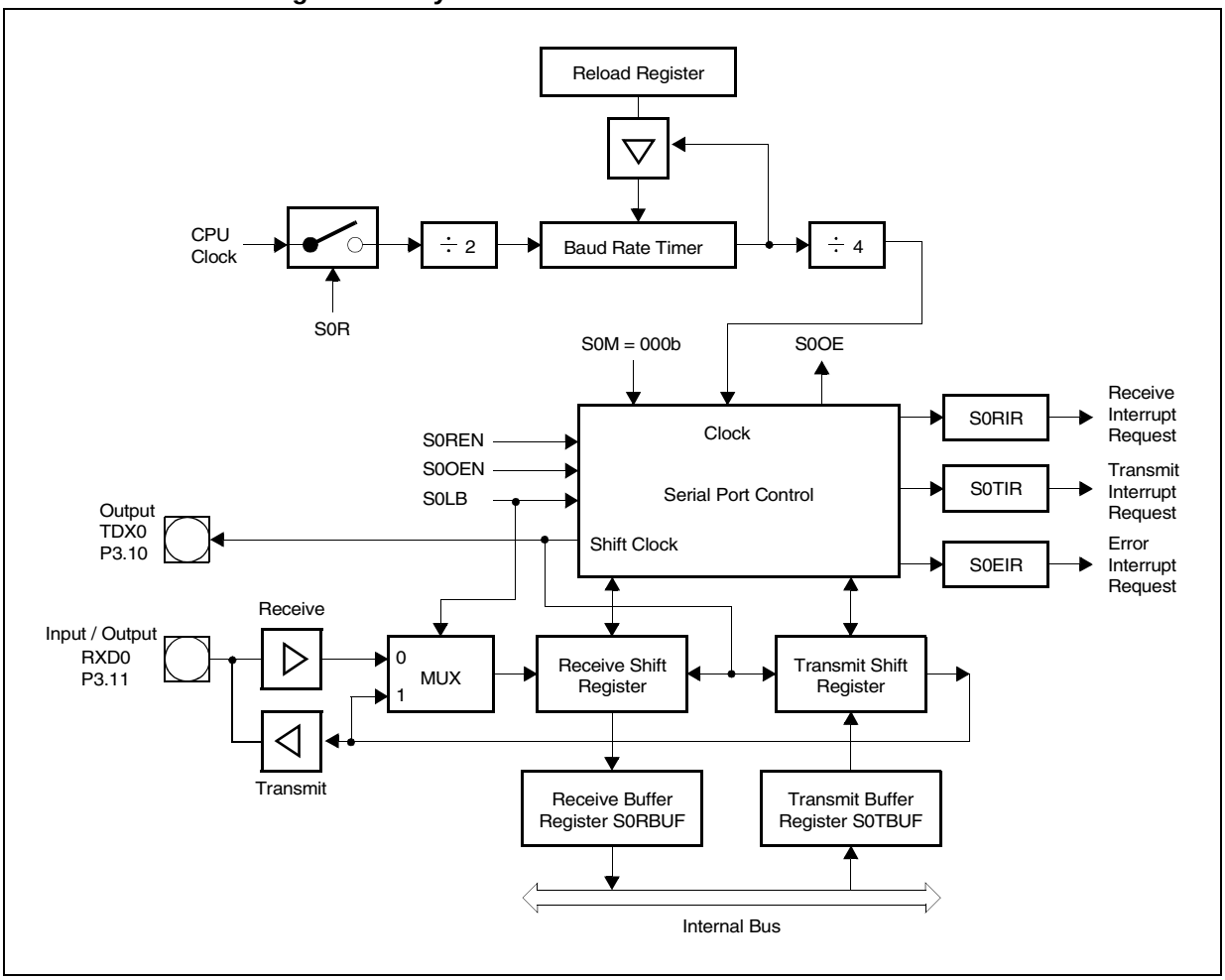
Note: In wake-up mode received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is ‘1’. If this bit is ‘0’, no receive interrupt request will be activated and no data will be transferred.

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

Figure 103. Synchronous mode of serial channel ASC0



Synchronous transmission begins within four CPU clock cycles after data has been loaded into S0TBUF, provided that S0R is set and S0REN = '0' (half-duplex, no reception). Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on. The data bits are transmitted synchronous with the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request flag S0TIR is set, and serial data transmission stops.

Pin TXD0/P3.10 must be configured for alternate data output, P3.10 = '1' and DP3.10 = '1', in order to provide the shift clock. Pin RXD0/P3.11 must also be configured for output (P3.11 = '1' and DP3.11 = '1') during transmission.

Synchronous reception is initiated by setting bit S0REN = '1'. If bit S0R = '1', the data applied at pin RXD0 are clocked into the receive shift register synchronous to the clock which is output at pin TXD0. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer S0RBUF, the receive interrupt request flag S0RIR is set, the receiver enable bit S0REN is reset, and serial data reception stops.

Pin TXD0/P3.10 must be configured for alternate data output, P3.10 = ‘1’ and DP3.10 = ‘1’, in order to provide the shift clock. Pin RXD0/P3.11 must be configured as alternate data input (DP3.11 = ‘0’).

Synchronous reception is stopped by clearing bit S0REN. A currently received byte is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register at the time the reception of the next byte is complete, both the error interrupt request flag S0EIR and the overrun error status flag S0OE will be set, if the overrun check has been enabled by S0OEN.

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 S0BG is written to. However, if S0R = ‘0’ at the time the write operation to S0BG is performed, the timer will not be reloaded until the first instruction cycle after S0R = ‘1’.

S0BG (FEB4h / 5Ah) SFR Reset Value: 0000h

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

Asynchronous mode baud rates

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

$$B_{Async} = \frac{f_{CPU}}{16 \times [2 + (S0BRS)] \times [(S0BRL) + 1]}$$

$$S0BRL = \left(\frac{f_{CPU}}{16 \times [2 + (S0BRS)] \times B_{Async}}\right) - 1$$

(S0BRL) represents the content of the reload register, taken as unsigned 13-bit integer, (S0BRS) represents the value of bit S0BRS (‘0’ or ‘1’), taken as integer.

Using the above equation, the maximum Baudrate can be calculated for any given clock speed. The device datasheet gives a table of values for Baudrate vs. reload register value for S0BRS = 0 and S0BRS = 1.

Synchronous mode baud rates

For synchronous operation, the Baud rate generator provides a clock with four times the rate of the established Baud rate. The Baud rate for synchronous operation of serial channel ASC0 can be determined by the following formula:

$$B_{Sync} = \frac{f_{CPU}}{4 \times [2 + (S0BRS)] \times [(S0BRL) + 1]}$$

$$S0BRL = \left(\frac{f_{CPU}}{4 \times [2 + (S0BRS)] \times B_{Sync}}\right) - 1$$

(S0BRL) represents the content of the reload register, taken as unsigned 13-bit integers, (S0BRS) represents the value of bit S0BRS (‘0’ or ‘1’), taken as integer.

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

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: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

Using the ASC0 Interrupts

For normal operation (besides the error interrupt) the ASC0 provides three interrupt requests to control data exchange via this serial channel:

- S0TBIRis activated when data is moved from S0TBUF to the transmit shift register.
- S0TIRis activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- S0RIRis activated when the received frame is moved to S0RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

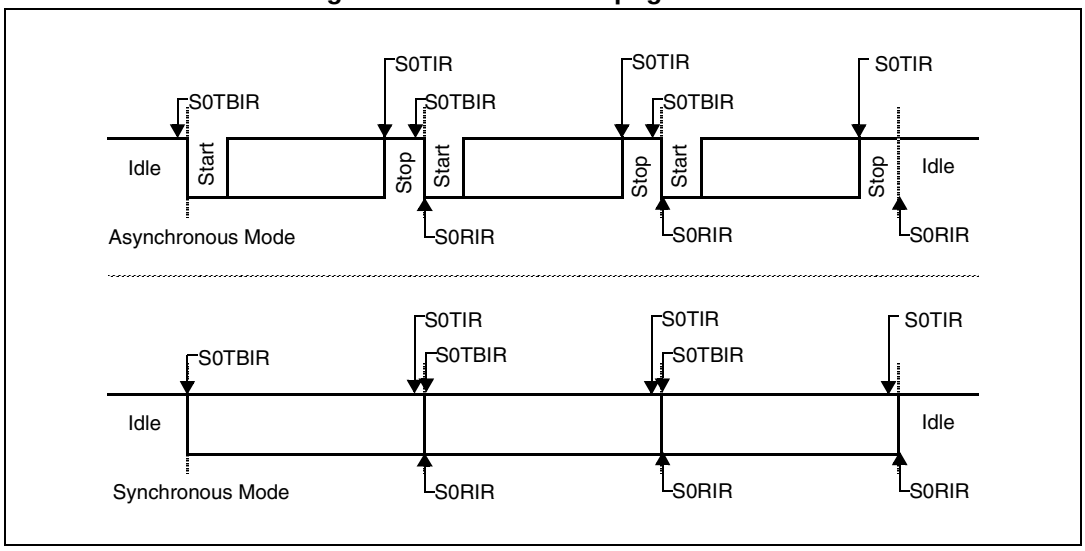
For single transfers is sufficient to use the transmitter interrupt (S0TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

For multiple back-to-back transfers it is necessary to load the following piece of data at last until the time the last bit of the previous frame has been transmitted. In asynchronous mode this leaves just one bit-time for the handler to respond to the transmitter interrupt request, in synchronous mode it is impossible at all.

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

As shown in the *Figure 104*, S0TBIR is an early trigger for the reload routine, while S0TIR indicates the completed transmission. Software using handshake therefore should rely on S0TIR at the end of a data block to make sure that all data has really been transmitted.

Figure 104. ASC0 interrupt generation



11 XBUS asynchronous / synchronous serial interface

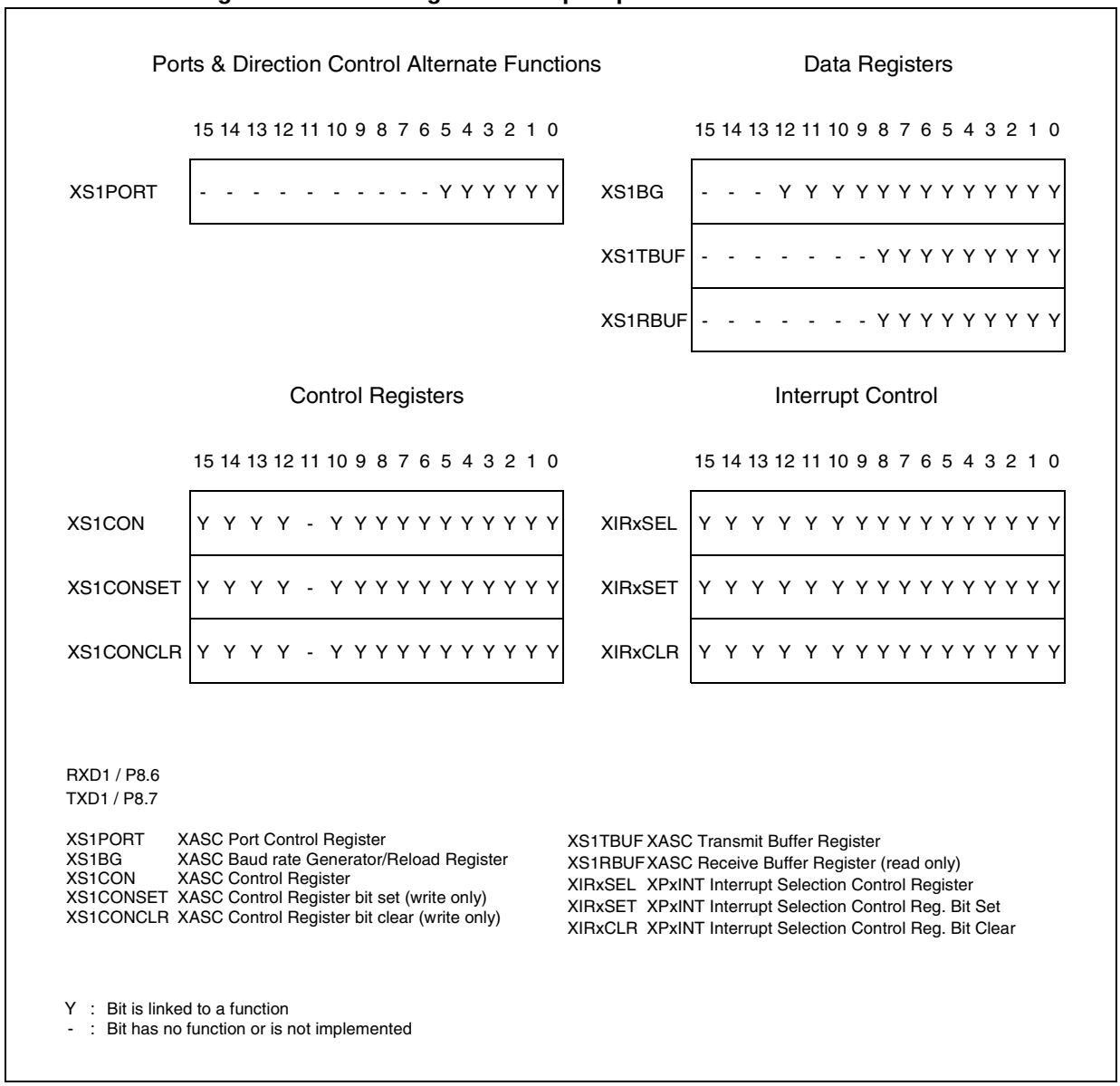
A second Asynchronous/Synchronous Serial Interface (XASC) is implemented on ST10F276. It is mapped on XBUS interface (Address range 00'E900h-00'E9FFh) and provides serial communication between the ST10F276 and other microcontrollers, microprocessors or external peripherals. The XASC is enabled by setting XPEN bit 2 of SYSCON register and bit 7 of XPERCON register.

In synchronous mode, data are transmitted or received synchronously to a shift clock which is generated by the ST10F276. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered.

For multiprocessor communication, a mechanism to distinguish address from data byte is included. Testing is supported by a loop-back option. A 13-bit Baud rate generator provides the XASC with a separate serial clock signal.

The main differences between ASC0 and XASC are restricted to the programming model and interrupt management, due to the constraints imposed by the XBUS with respect to the standard ST10 peripheral bus (registers are not bit addressable, XBUS interrupt channels sharing with other X-Peripherals). In terms of general functionality and performance, the two modules are completely equivalent.

Figure 105. XBUS registers and port pins associated with XASC



The operating mode of the serial channels XASC is controlled by its control register XS1CON. Being an XBUS register, it is not bit-addressable: for this reason, a couple of additional XBUS registers are provided to set and clear single bits of XS1CON emulating the bit-addressability (see XS1CONSET, XS1CONCLR. This register contains control bits for mode and error check selection, and status flags for error identification.

XS1CON (E900h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S1R	S1LB	S1BRS	S1ODD	-	**S1OE**	**S1FE**	**S1PE**	S1OEN	S1FEN	S1PEN	S1REN	S1STP	S1M		
RW	RW	RW	RW		RW	RW	RW	RW	RW	RW	RW	RW	RW		

Bit	Function
S1M	**XASC Mode Control** 0 0 0: 8-bit data synchronous operation 0 0 1: 8-bit data asynchronous operation 0 1 0: Reserved. Do not use this combination 0 1 1: 7-bit data + parity asynchronous operation 1 0 0: 9-bit data asynchronous operation 1 0 1: 8-bit data + wake up bit asynchronous operation 1 1 0: Reserved. Do not use this combination 1 1 1: 8-bit data + parity asynchronous operation
S1STP	**Number of Stop bit Selection asynchronous operation** 0: One stop bit 1: Two stop bit
S1REN	**Receiver Enable bit** 0: Receiver disabled 1: Receiver enabled (Reset by hardware after reception of byte in synchronous mode)
S1PEN	**Parity Check Enable bit** asynchronous operation 0: Ignore parity 1: Check parity
S1FEN	**Framing Check Enable bit** asynchronous operation 0: Ignore framing errors 1: Check framing errors
S1OEN	**Overrun Check Enable bit** 0: Ignore overrun errors 1: Check overrun errors
S1PE	**Parity Error Flag** Set by hardware on a parity error (S1PEN = ‘1’). Must be reset by software.
S1FE	**Framing Error Flag** Set by hardware on a framing error (S1FEN = ‘1’). Must be reset by software.
S1OE	**Overrun Error Flag** Set by hardware on an overrun error (S1OEN = ‘1’). Must be reset by software.
S1ODD	**Parity Selection bit** 0: Even parity (parity bit set on odd number of ‘1’s in data) 1: Odd parity (parity bit set on even number of ‘1’s in data)
S1BRS	**Baud rate Selection bit** 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3rd
S1LB	**Loopback Mode Enable bit** 0: Standard transmit/receive mode 1: Loopback mode enabled
S1R	**Baud rate Generator Run bit** 0: Baud rate generator disabled (XASC inactive) 1: Baud rate generator enabled

XS1CONSET (E902h) XBUS Reset Value: 0000h

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

Bit	Function
SET.y	Writing a ‘1’ will set the corresponding bit in XS1CON register. Writing a ‘0’ has no effect.

XS1CONCLR (E904h) XBUS Reset Value: 0000h

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

Bit	Function
CLR.y	Writing a ‘1’ will clear the corresponding bit in XS1CON register. Writing a ‘0’ has no effect.

A transmission is started by writing to the Transmit Buffer register XS1TBUF (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 XS1TBUF 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 S1REN. 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 XS1RBUF.

Bit in the upper half of XS1RBUF which are not valid in the selected operating mode will be read as zeros.

Data reception is double-buffered, so 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 S1OEN. When enabled, the overrun error status flag S1OE and the error interrupt request flag 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 S1LB) 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 Port8 pins are not necessary.

Note: Serial data transmission or reception is only possible when the Baud rate Generator Run bit S1R is set to '1'. Otherwise the serial interface is idle.
Do not program the mode control field S1M in register XS1CON to one of the reserved combinations to avoid unpredictable behavior of the serial interface.

XS1TBUF (E908h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	Transmit Data Buffer								
							RW								

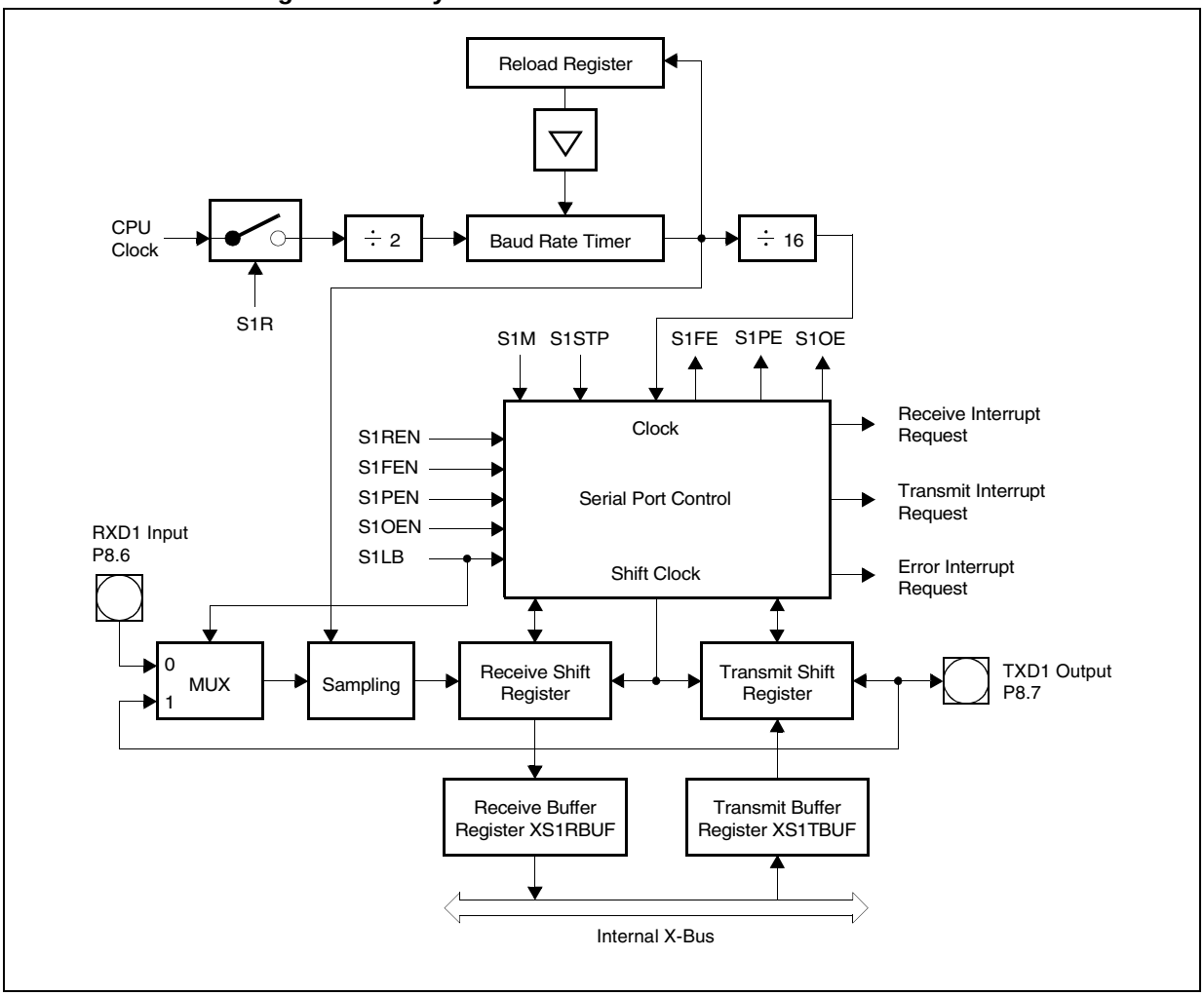
XS1RBUF (E90Ah) XBUS Reset Value: 00xxh

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

11.1 Asynchronous operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same Baud rate. Data is transmitted on pin TXD1/P8.7 and received on pin RXD1/P8.6. These signals are alternate functions of Port8 pins.

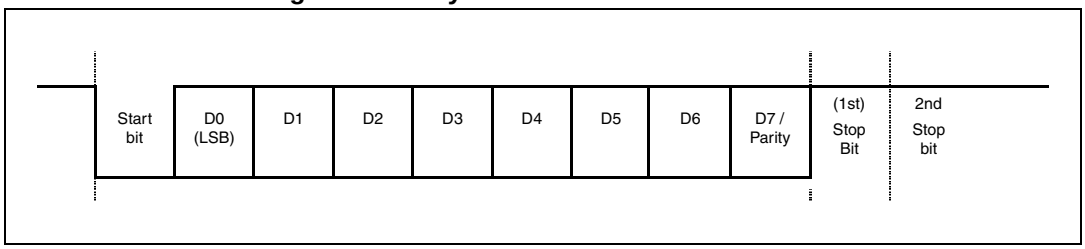
Figure 106. Asynchronous mode of serial channel XASC



Asynchronous data frames

8-bit data frames either consist of 8 data bits D7...D0 (S1M = '001b'), or of 7 data bits D6...D0 plus an automatically generated parity bit (S1M = '011b'). Parity may be odd or even, depending on bit S1ODD in register XS1CON. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S1PEN (always OFF in 8-bit data mode). The parity error flag S1PE 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 XS1RBUF.7.

Figure 107. Asynchronous 8-bit data frames



9-bit data frames either consist of 9 data bits D8...D0 (S1M = ‘100b’), of 8 data bits D7...D0 plus an automatically generated parity bit (S1M = ‘111b’) or of 8 data bit D7...D0 plus wake-up bit (S1M = ‘101b’). Parity may be odd or even, depending on bit S1ODD in register XS1CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is ‘1’. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S1PEN (always OFF in 9-bit data and wake-up mode). The parity error flag S1PE 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 XS1RBUF.

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 S1M.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 102*).

Asynchronous transmission begins at the next overflow of the divide-by-16 counter (see *Figure 102*), provided that S1R is set and data has been loaded into XS1TBUF. The transmitted data frame consists of three basic elements:

- the start bit,
- the data field (8 or 9 bits, LSB first, including a parity bit, if selected),
- the delimiter (1 or 2 stop bits).

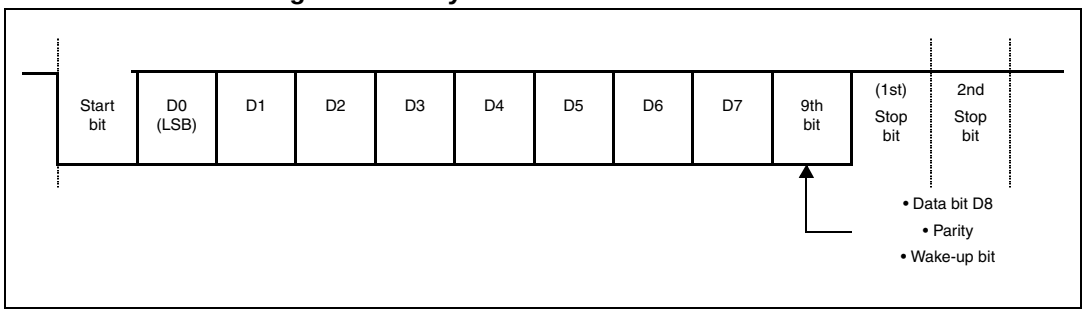
Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into XS1TBUF is immediately moved to the transmit shift register thus freeing XS1TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag being set. XS1TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag 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 TXD1/P8.7 must be configured for alternate data output, P8.7 = ‘1’ and DP8.7 = ‘1’.

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD1, provided that bit S1R and S1REN are set. The receive data input pin RXD1 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 RXD1. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

Figure 108. Asynchronous 9-bit data frames



When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register XS1RBUF. Simultaneously, the receive interrupt request flag 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 RXD1/P8.6 must be configured for input, using direction control register DP8.6 = '0'.

Asynchronous reception is stopped by clearing bit S1REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bit that follow this frame will not be recognized.

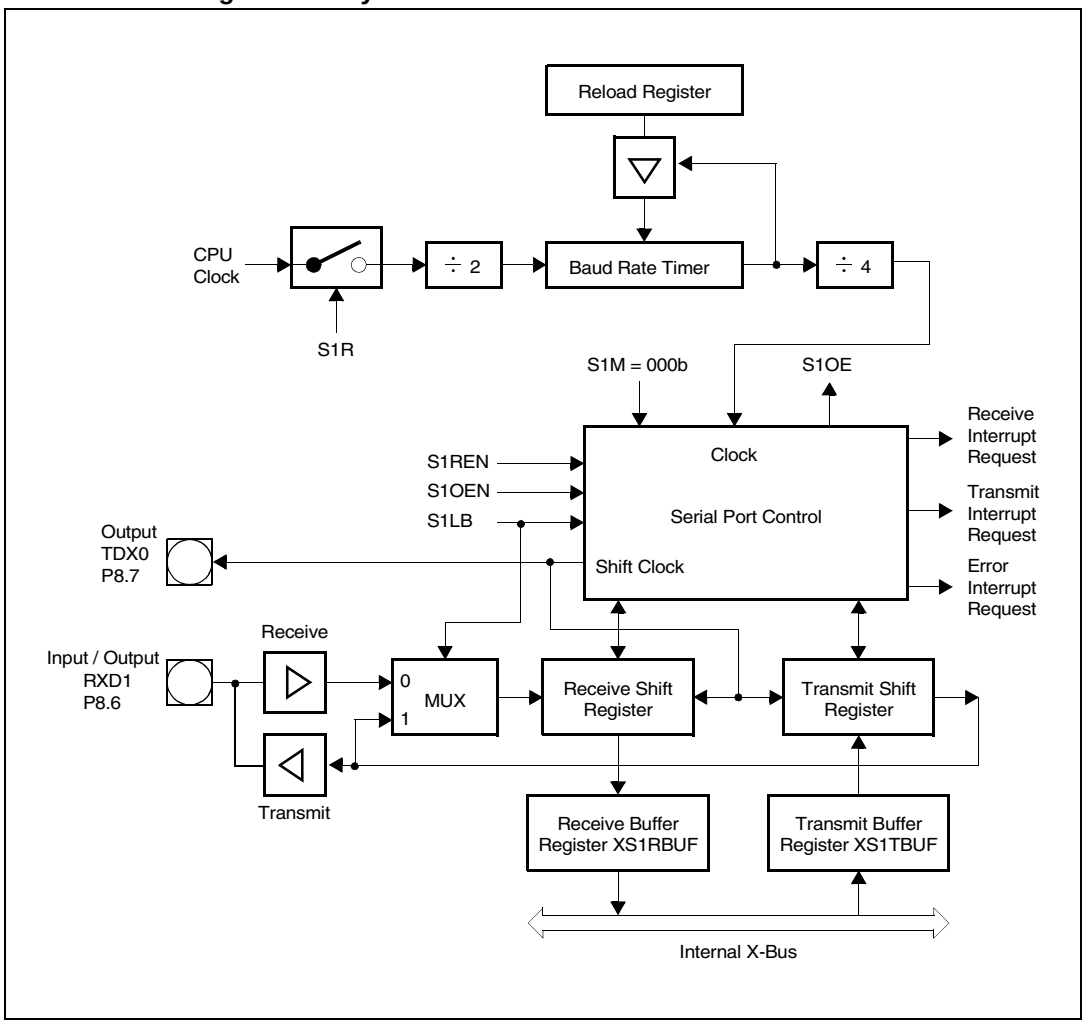
Note: In wake-up mode received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is '1'. If this bit is '0', no receive interrupt request will be activated and no data will be transferred.

11.2 Synchronous operation

Synchronous mode supports half-duplex communication, basically for simple I/O expansion via shift registers. Data is transmitted and received via pin RXD1/P8.6, while pin TXD1/P8.7 outputs the shift clock. These signals are alternate functions of Port8 pins. Synchronous mode is selected with S1M = '000b'.

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

Figure 109. Synchronous mode of serial channel XASC



Synchronous transmission begins within four CPU clock cycles after data has been loaded into XS1TBUF, provided that S1R is set and S1REN = ‘0’ (half-duplex, no reception). Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into XS1TBUF is immediately moved to the transmit shift register thus freeing XS1TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag being set. XS1TBUF may now be loaded with the next data, while transmission of the previous one is still going on. The data bits are transmitted synchronous with the shift clock. After the bit time for the 8th data bit, both pins TXD1 and RXD1 will go high, the transmit interrupt request flag is set, and serial data transmission stops.

Pin TXD1/P8.7 must be configured for alternate data output, P8.7 = ‘1’ and DP8.7 = ‘1’, in order to provide the shift clock. Pin RXD1/P8.6 must also be configured for output (P8.6 = ‘1’ and DP8.6 = ‘1’) during transmission.

Synchronous reception is initiated by setting bit S1REN = ‘1’. If bit S1R = 1, the data applied at pin RXD1 are clocked into the receive shift register synchronous to the clock which is output at pin TXD1. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer XS1RBUF, the receive interrupt request flag is set, the receiver enable bit S1REN is reset, and serial data reception stops.

Pin TXD1/P8.7 must be configured for alternate data output, P8.7 = ‘1’ and DP8.7 = ‘1’, in order to provide the shift clock. Pin RXD1/P8.6 must be configured as alternate data input (DP8.6 = ‘0’).

Synchronous reception is stopped by clearing bit S1REN. 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 and the overrun error status flag S1OE will be set, if the overrun check has been enabled by S1OEN.

11.3 Hardware error detection

To improve the safety of serial data exchange, the serial channel XASC provides an error interrupt request flag, which indicates the presence of an error, and three (selectable) error status flags in register XS1CON, which indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request flag will be set simultaneously with the receive interrupt request flag, if one or more of the following conditions are met:

- If the framing error detection enable bit S1FEN is set and any of the expected stop bit is not high, the framing error flag S1FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).
- If the parity error detection enable bit S1PEN is set in parity bit receive modes, and the parity check on the received data bit proves false, the parity error flag S1PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).
- If the overrun error detection enable bit S1OEN 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 S1OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

11.4 XASC baud rate generation

The serial channel XASC 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 S1R in register XS1CON. 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 S1BRS. If S1BRS = ‘1’, the clock signal is additionally divided to 2/3rd of its frequency (see formulas and table). So the Baud rate of XASC is determined by the CPU clock, the reload value, the value of S1BRS and the operating mode (asynchronous or synchronous).

Register XS1BG is the dual-function Baud rate Generator/Reload register. Reading XS1BG returns the content of the timer (bit 15...13 return zero), while writing to XS1BG 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 XS1BG is written to. However, if S1R = '0' at the time the write operation to XS1BG is performed, the timer will not be reloaded until the first instruction cycle after S1R = '1'.

XS1BG (E906h) XBUS Reset Value: 0000h

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

Asynchronous mode baud rates

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

$$B_{Async} = \frac{f_{CPU}}{16 \times [2 + (S1BRS)] \times [(S1BRL) + 1]}$$

$$S1BRL = \left(\frac{f_{CPU}}{16 \times [2 + (S1BRS)] \times B_{Async}}\right) - 1$$

(S1BRL) represents the content of the reload register, taken as unsigned 13-bit integer, (S1BRS) represents the value of bit S1BRS ('0' or '1'), taken as integer.

Using the above equation, the maximum Baudrate can be calculated for any given clock speed. The device datasheet gives a table of values for Baudrate vs. reload register value for S1BRS = 0 and S1BRS = 1.

Synchronous mode baud rates

For synchronous operation, the Baud rate generator provides a clock with four times the rate of the established Baud rate. The Baud rate for synchronous operation of serial channel XASC can be determined by the following formula:

$$B_{Sync} = \frac{f_{CPU}}{4 \times [2 + (S1BRS)] \times [(S1BRL) + 1]}$$

$$S1BRL = \left(\frac{f_{CPU}}{4 \times [2 + (S1BRS)] \times B_{Sync}}\right) - 1$$

(S1BRL) represents the content of the reload register, taken as unsigned 13-bit integers, (S1BRS) represents the value of bit S1BRS ('0' or '1'), taken as integer.

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

11.5 XASC interrupt control

Up to four interrupt control registers (XIRxSEL, x = 0, 1, 2, 3) are provided in order to select the source of the XBUS interrupt: the transmit interrupt, the transmit buffer interrupt, the receive interrupt and the error interrupt of serial channel XASC are linked to one of the

XPxIC registers (x = 0, 1, 2, 3). In particular, the four interrupt lines are available on the following interrupt vectors:

- ReceiveXP0INTXP1INTXP2INT
- TransmitXP0INTXP1INTXP2INT
- Transmit BufferXP0INTXP1INTXP2INT
- ErrorXP3INT

Refer to *Section 5.7: X-peripheral interrupt on page 117* for details.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register XS1CON.

Note: The error status flags S1FE / S1PE / S1OE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

Using the XASC interrupts

For normal operation (besides the error interrupt) the XASC provides three interrupt requests to control data exchange via this serial channel:

- Transmit Buffer (TBIR) internal interrupt signal is activated when data is moved from XS1TBUF to the transmit shift register.
- Transmit (TIR) internal interrupt signal is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- Receive (RIR) internal interrupt signal is activated when the received frame is moved to XS1RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

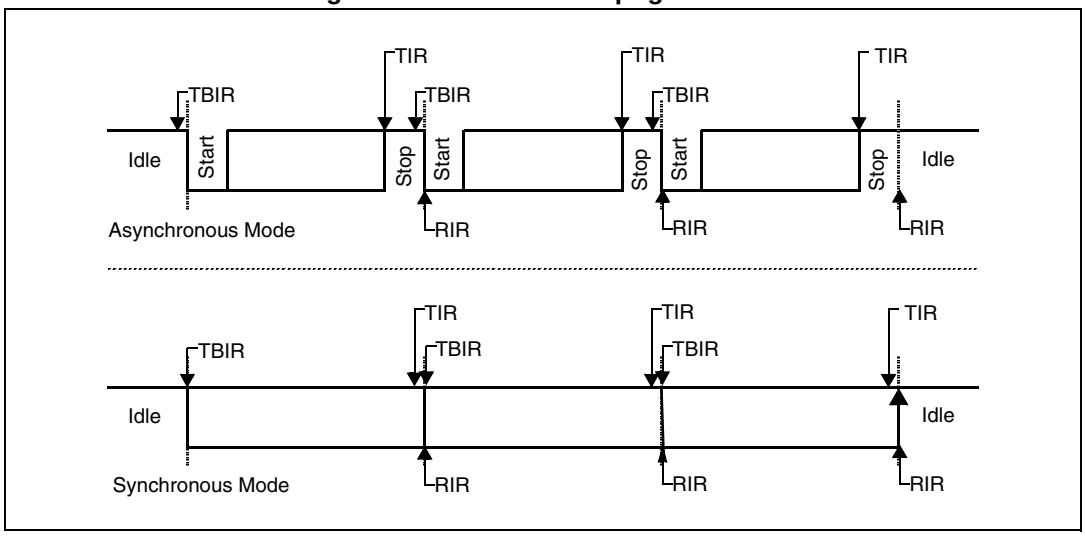
For single transfers it is sufficient to use the transmitter interrupt (TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

For multiple back-to-back transfers it is necessary to load the following piece of data at last until the time the last bit of the previous frame has been transmitted. In asynchronous mode this leaves just one bit-time for the handler to respond to the transmitter interrupt request, in synchronous mode it is impossible at all.

Using the transmit buffer interrupt (TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as XS1TBUF may be reloaded while the previous data is still being transmitted.

As shown in the *Figure 104*, TBIR is an early trigger for the reload routine, while TIR indicates the completed transmission. Software using handshake therefore should rely on TIR at the end of a data block to make sure that all data has really been transmitted.

Figure 110. XASC interrupt generation



12 High-speed synchronous serial interface

The high-speed synchronous serial interface SSC provides flexible high-speed serial communication between the ST10F276 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 multi-master interconnections or operate like the popular SPI interface. It can communicate with shift registers (I/O expansion), peripherals (such as EEPROMs) 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 111. SFRs and port pins associated with the SSC

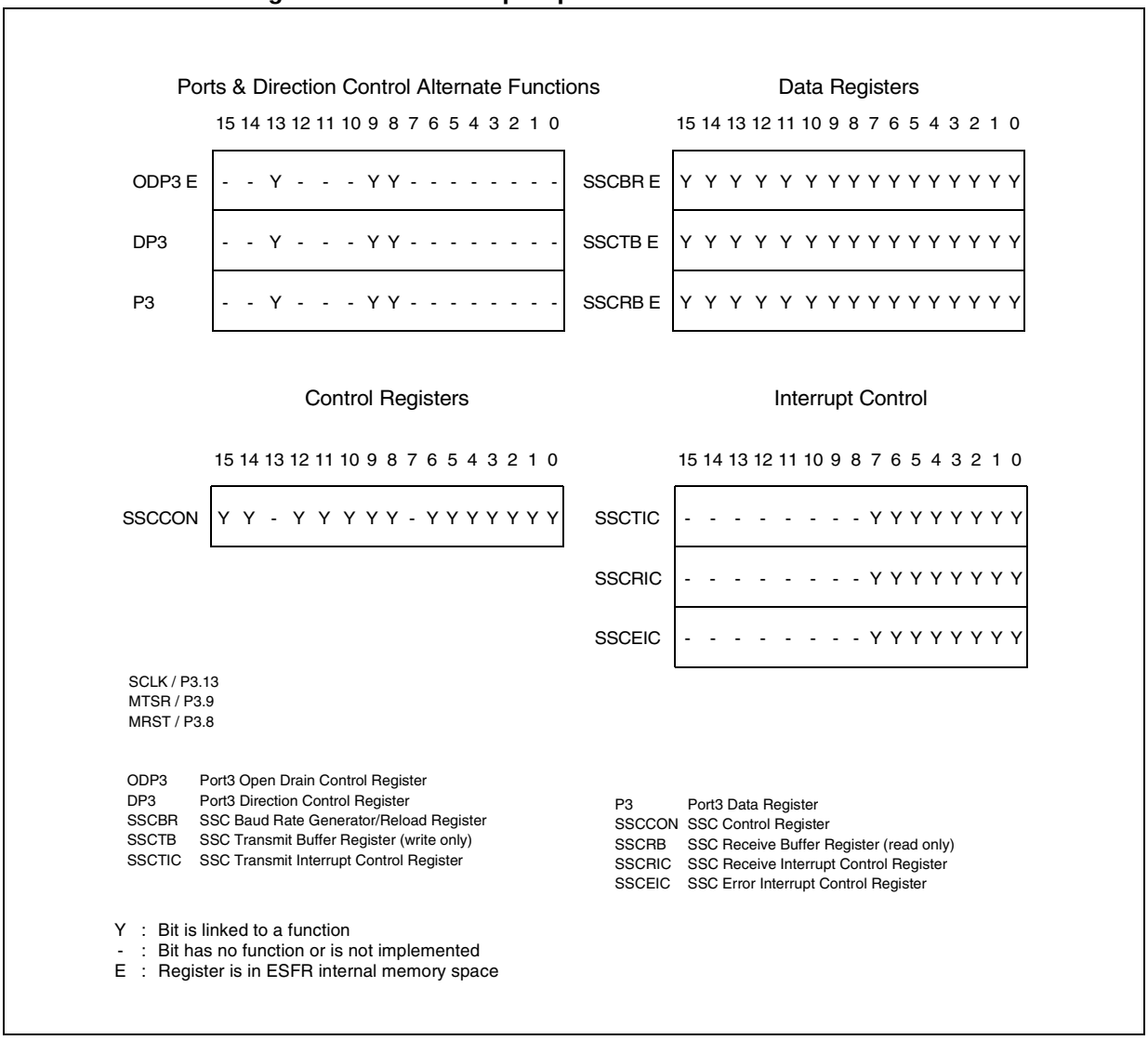
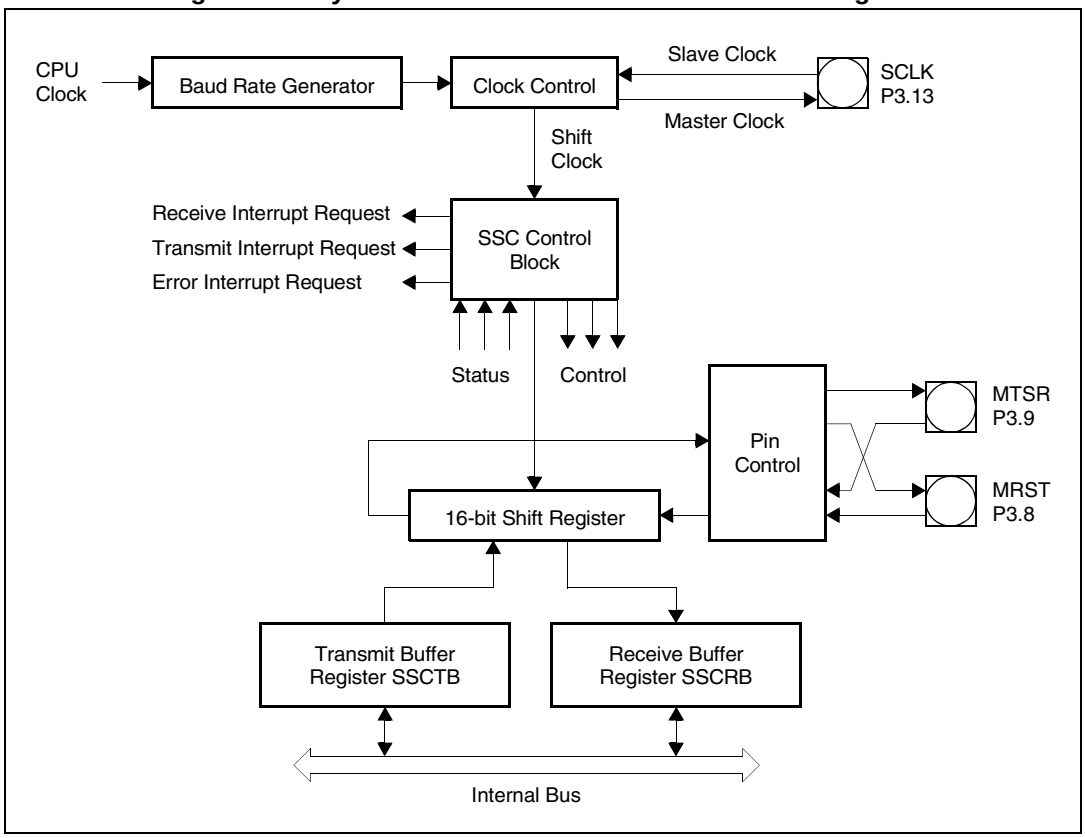


Figure 112. Synchronous serial channel SSC block diagram



The operating mode of the serial channel SSC is controlled by its bit-addressable control register SSCCON. This register serves for two purposes:

- During programming (SSC disabled by SSCEN = '0') it provides access to a set of control bit.
- During operation (SSC enabled by SSCEN = '1') it provides access to a set of status flags. Register SSCCON is shown below in each of the two modes.

SSCRB (F0B2h / 59h) ESFR Reset Value: xxxxh

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

SSCTB (F0B0h / 58h) ESFR Reset Value: 0000h

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

SSCCON (FFB2h / D9h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN=0	SSC MS	-	SSC AREN	SSC BEN	SSC PEN	SSC REN	SSC TEN	-	SSC PO	SSC PH	SSC HB	SSCBM			
RW	RW		RW	RW	RW	RW	RW		RW	RW	RW	RW			

Bit	Function (programming mode, SSCEN = '0')
SSCBM	**SSC Data Width Selection** 0: Reserved. Do not use this combination. 1...15: Transfer Data Width is 2...16-bit [(SSCBM)+1]
SSCHB	**SSC Heading Control bit** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
SSCPH	**SSC Clock Phase Control bit** 0: Shift transmit data on the leading clock edge, latch on trailing edge 1: Latch receive data on leading clock edge, shift on trailing edge
SSCPO	**SSC Clock Polarity Control bit** 0: Idle clock line is low, leading clock edge is low-to-high transition 1: Idle clock line is high, leading clock edge is high-to-low transition
SSCTEN	**SSC Transmit Error Enable bit** 0: Ignore transmit errors 1: Check transmit errors
SSCREN	**SSC Receive Error Enable bit** 0: Ignore receive errors 1: Check receive errors
SSCPEN	**SSC Phase Error Enable bit** 0: Ignore phase errors 1: Check phase errors
SSCBEN	**SSC Baudrate Error Enable bit** 0: Ignore baudrate errors 1: Check baudrate errors
SSCAREN	**SSC Automatic Reset Enable bit** 0: No additional action upon a baudrate error 1: The SSC is automatically reset upon a baudrate error
SSCMS	**SSC Master Select bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	SSC Enable bit = '0' Transmission and reception disabled. Access to control bits.

SSCCON (FFB2h / D9h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN=1	SSC MS	-	SSC BSY	SSC BE	SSC PE	SSC RE	SSC TE	-	-	-	-	SSCBC			
RW	RW		RW	RW	RW	RW	RW					RW			

Bit	Function (operating mode, SSCEN = '1')
SSCBC	**SSC bit Count Field** Shift counter is updated with every shifted bit. **Do not write to**
SSCTE	**SSC Transmit Error Flag** 1: Transfer starts with the slave's transmit buffer not being updated
SSCRE	**SSC Receive Error Flag** 1: Reception completed before the receive buffer was read
SSCPE	**SSC Phase Error Flag** 1: Received data changes around sampling clock edge
SSCBE	**SSC Baud rate Error Flag** 1: More than factor 2 or 0.5 between Slave's actual and expected Baud rate
SSCBSY	**SSC Busy Flag:** Set while a transfer is in progress. **Do not write to**
SSCMS	**SSC Master Select bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable bit = '1'** Transmission and reception enabled. Access to status flags and M/S control.

Note: The target of an access to SSCCON (control bit or flags) is determined by the state of SSCEN prior to the access. Writing C057h to SSCCON in programming mode (SSCEN = '0') will initialize the SSC (SSCEN was '0') and then turn it on (SSCEN = '1'). When writing to SSCCON, make sure that reserved locations receive zeros.

The shift register of the SSC is connected to both the transmit pin and the receive pin via the pin control logic (see *Figure 112*). Transmission and reception of serial data is synchronized and takes place at the same time, so the same number of transmitted bit is also received. Transmit data is written into the Transmit Buffer SSCTB. It is moved to the shift register as soon as this is empty. An SSC-master (SSCMS = '1') immediately begins transmitting, while an SSC-slave (SSCMS = '0') will wait for an active shift clock. When the transfer starts, the busy flag SSCBSY is set and a transmit interrupt request (SSCTIR) will be generated to indicate that SSCTB may be reloaded again. When the programmed number of bit (2...16) has been transferred, the contents of the shift register are moved to the Receive Buffer SSCRB and a receive interrupt request (SSCRIR) will be generated. If no further transfer is to take place (SSCTB is empty), SSCBSY will be cleared at the same time. Software should not modify SSCBSY, as this flag is hardware controlled. Only one SSC can be master at a given time.

The transfer of serial data bit can be programmed in the following ways:

- The data width can be chosen from 2 bits to 16 bits.
- Transfer may start with the LSB or the MSB.
- The shift clock may be idle low or idle high.
- Data bit may be shifted with the leading or trailing edge of the clock signal.
- The Baud rate may be set for a range of values (refer to *Section 12.3: Baud rate generation on page 280* for the formula to calculate values or to the device datasheet for specific values).
- The shift clock can be generated (master) or received (slave).

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

The data width selection supports the transfer of frames of any length, from 2 bit “characters” up to 16 bit “characters”. Starting with the LSB (SSCHB = ‘0’) allows communication with ASC0 devices in synchronous mode like serial interfaces. Starting with the MSB (SSCHB = ‘1’) allows operation compatible with the SPI interface.

Regardless which data width is selected and whether the MSB or the LSB is transmitted first, the transfer data is always right aligned in registers SSCTB and SSCRB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of SSCTB are ignored, the unselected bits of SSCRB will be not valid and should be ignored by the receiver service routine.

The clock control allows the adaptation of transmit and receive behavior of the SSC to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out transmit data, while the other clock edge is used to latch in receive data. Bit SSCPH selects the leading edge or the trailing edge for each function. Bit SSCPO selects the level of the clock line in the idle state. So for an idle-high clock the leading edge is a falling one, a 1-to-0 transition. *Figure 113 on page 276* is a summary.

12.1 Full-duplex operation

The different devices are connected through three lines. The definition of these lines is always determined by the master: The line connected to the master's data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode (DP3.13 = ‘0’). The output of the master’s shift register is connected to the external transmit line, which in turn is connected to the slaves’ shift register input.

The output of the slaves’ shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins is determined by the master or slave operation of the individual device.

Note: The shift direction shown in the Figure 113 applies for MSB-first operation as well as for LSB-first operation.

When initializing the devices in this configuration, select one device for master operation (SSCMS = ‘1’), all others must be programmed for slave operation (SSCMS = ‘0’). Initialization includes the operating mode of the device's SSC and also the function of the respective port lines (see *Section 12.2.1: Port control on page 279*).

Figure 113. Serial clock phase and polarity options

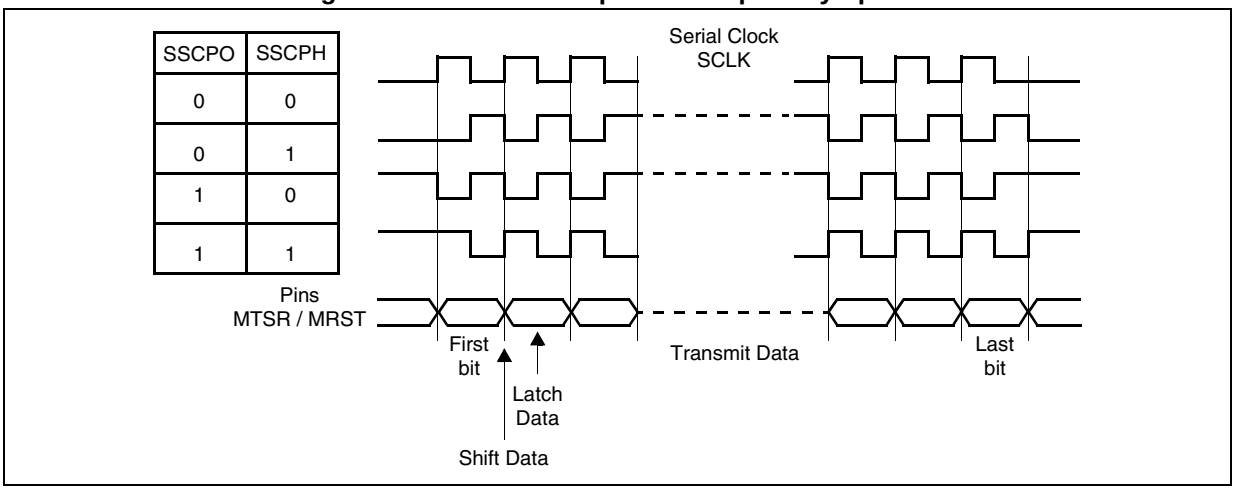
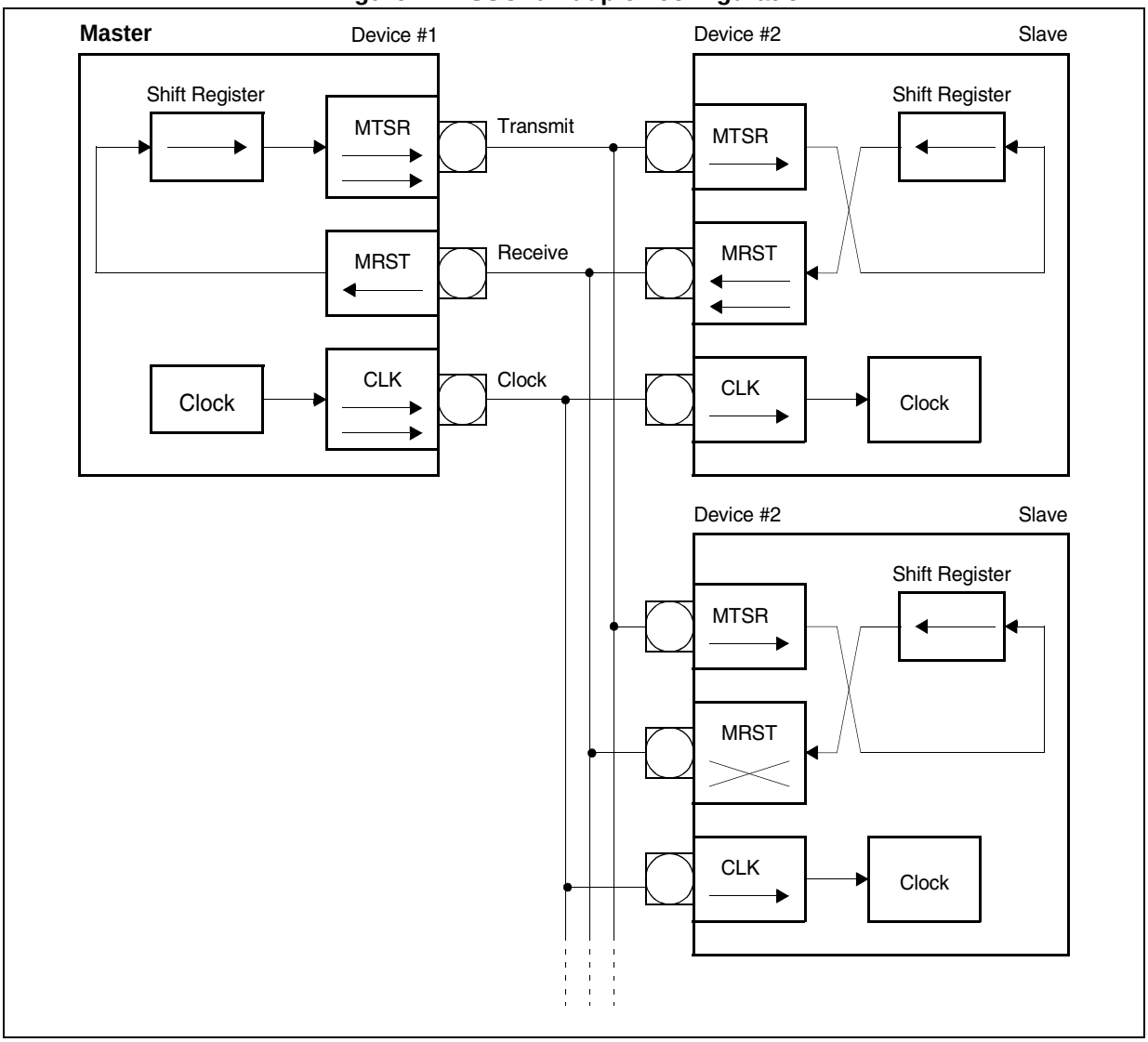


Figure 114. SSC full duplex configuration



The data output pins MRST of all slave devices are connected together onto the one receive line in this configuration. During a transfer each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

Only one slave drives the line, it enables the driver of its MRST pin. All the other slaves have to program there MRST pins to input. So only one slave can put its data onto the master's receive line. Only receiving of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output, until it gets a de-selection signal or command.

The slaves use open drain output on MRST. This forms a AND-wired connection. The receive line needs an external pull-up in this case. Corruption of the data on the receive line sent by the selected slave is avoided, when all slaves which are not selected for transmission to the master only send ones (‘1’). Since this high level is not actively driven onto the line, but only held through the pull-up device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initialization of the SSC, the serial interfaces can be enabled. For a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either '0' or '1', until the first transfer will start. After a transfer the alternate data line will always remain at the logic level of the last transmitted data bit.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register SSCTB. This value is copied into the shift register (which is assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock from the Baud rate generator (transmission only starts, if SSCEN = ‘1’). Depending on the selected clock phase, also a clock pulse will be generated on the SCLK line.

With the opposite clock edge the master at the same time latches and shifts in the data detected at its input line MRST. This “exchanges” the transmit data with the receive data. Since the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master's shift register, shifting out the data contained in the registers, and shifting in the data detected at the input line. After the pre-programmed number of clock pulses (via the data width selection) the data transmitted by the master is contained in all slaves’ shift registers, while the master's shift register holds the data of the selected slave. In the master and all slaves the content of the shift register is copied into the receive buffer SSCRB and the receive interrupt flag SSCRIR is set.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST, when the content of the transmit buffer is copied into the slave's shift register. It will not wait for the next clock from the Baud rate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock edge generated by the master may be already used to clock in the first data bit. So the slave's first data bit must already be valid at this time.

*Note: A transmission **and** a reception takes place at the same time, regardless whether valid data has been transmitted or received. This is different from asynchronous reception on ASC0.*

The initialization of the SCLK pin on the master requires some attention in order to avoid undesired clock transitions, which may disturb the other receivers. The state of the internal alternate output lines is '1' as long as the SSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the SSC with an idle-low clock

(SSCPO = ‘0’) will drive the alternate data output and (via the AND) the port pin SCLK immediately low. To avoid this, use the following sequence:

- Select the clock idle level (SSCPO = ‘x’)
- Load the port output latch with the desired clock idle level (P3.13 = ‘x’)
- Switch the pin to output (DP3.13 = ‘1’)
- Enable the SSC (SSCEN = ‘1’)
- If SSCPO = ‘0’: enable alternate data output (P3.13 = ‘1’)

The same mechanism as for selecting a slave for transmission (separate select lines or special commands) may also be used to move the role of the master to another device in the network. In this case the previous master and the future master (previous slave) will have to toggle their operating mode (SSCMS) and the direction of their port pins (see description above).

12.2 Half duplex operation

In a half duplex configuration only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both pins MTSR and MRST of each device, the clock line is connected to the SCLK pin.

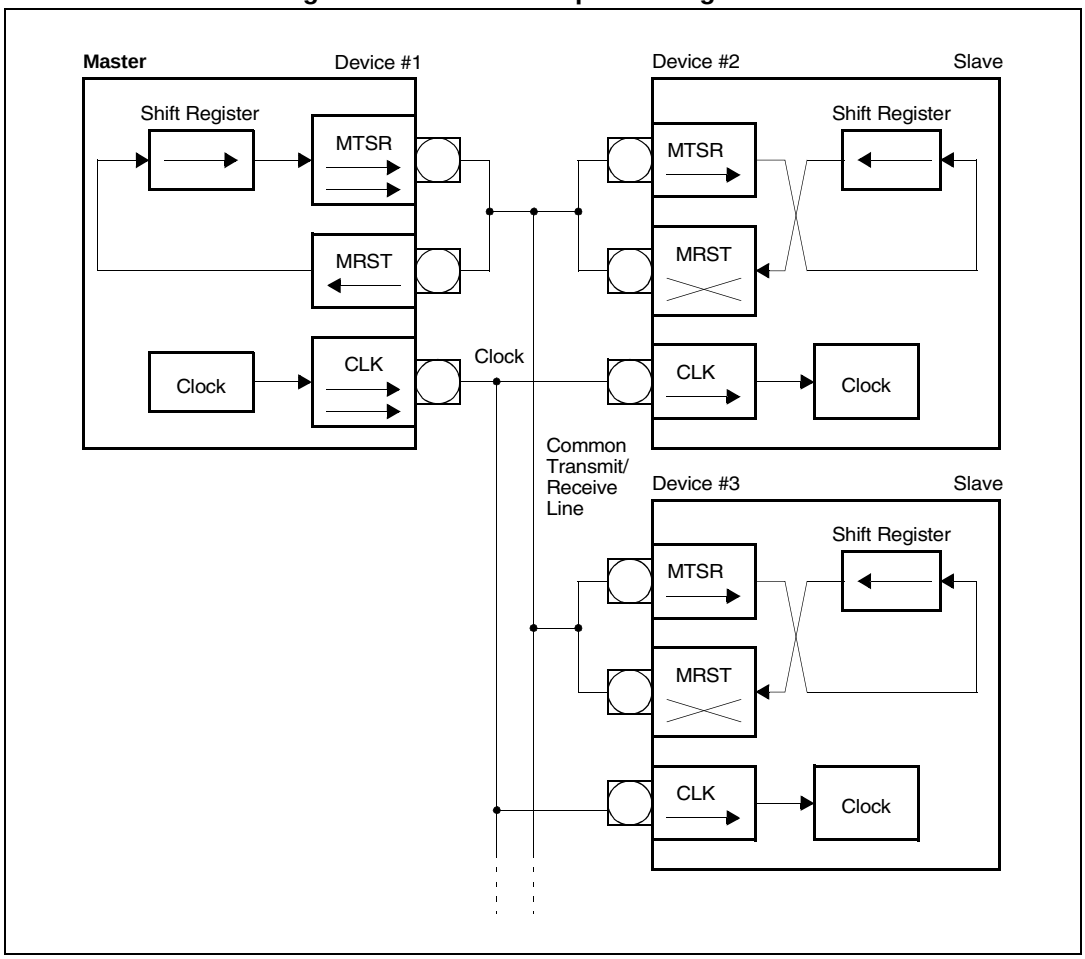
The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

Similar to full duplex mode there are two ways to avoid collisions on the data exchange line:

- Only the transmitting device may enable its transmit pin driver
- The non-transmitting devices use open drain output and only send ones.

Since the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By these means any corruptions on the common data exchange line are detected, where the received data is not equal to the transmitted data.

Figure 115. SSC half duplex configuration



Continuous transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCTB is empty and ready to be loaded with the next transmit data. If SSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line there is no gap between the two successive frames, so two bytes transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. It is just a matter of software, how long a total data frame length can be. This option can also be used to interface to byte wide and word wide devices on the same serial bus.

Note: Of course, this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.

12.2.1 Port control

The SSC uses three pins of Port3 to communicate with the external world. Pin P3.13/SCLK serves as the clock line, while pins P3.8/MRST (Master Receive / Slave Transmit) and P3.9/MTSR (Master Transmit / Slave Receive) serve as the serial data input/output lines. The operation of these pins depends on the selected operating mode (master or slave). In order to enable the alternate output functions of these pins instead of the general purpose

I/O operation, the respective port latches have to be set to '1', since the port latch outputs and the alternate output lines are ANDed. When an alternate data output line is not used (function disabled), it is held at a high level, allowing I/O operations via the port latch. The direction of the port lines depends on the operating mode. The SSC will automatically use the correct alternate input or output line of the ports when switching modes. The direction of the pins, however, must be programmed by the user, as shown in the tables.

Using the open drain output feature helps to avoid bus contention problems and reduces the need for hardwired hand-shaking or slave select lines. In this case it is not always necessary to switch the direction of a port pin. The table below summarizes the required values for the different modes and pins.

Pin	Master mode			Slave mode		
	Function	Port latch	Direction	Function	Port latch	Direction
P3.13 / SCLK	Serial Clock Output	P3.13 = ‘1’	DP3.13 = ‘1’	Serial Clock Input	P3.13 = ‘x’	DP3.13 = ‘0’
P3.9 / MTSR	Serial Data Output	P3.9 = ‘1’	DP3.9 = ‘1’	Serial Data Input	P3.9 = ‘x’	DP3.9 = ‘0’
P3.8 / MRST	Serial Data Input	P3.8 = ‘x’	DP3.8 = ‘0’	Serial Data Output	P3.8 = ‘1’	DP3.8 = ‘1’

Note: In the table above, an 'x' means that the actual value is irrelevant in the respective mode, however, it is recommended to set these bits to '1', so they are already in the correct state when switching between master and slave mode.

12.3 Baud rate generation

The serial channel SSC has its own dedicated 16-bit Baud rate generator with 16-bit reload capability, allowing Baud rate generation independent from the timers.

The Baud rate generator is clocked by $f_{CPU}/2$. The timer is counting downwards and can be started or stopped through the global enable bit SSCEN in register SSCCON. Register SSCBR is the dual-function Baud Rate Generator/Reload register. Reading SSCBR, while the SSC is enabled, returns the content of the timer. Reading SSCBR, while the SSC is disabled, returns the programmed reload value. In this mode the desired reload value can be written to SSCBR.

Note: Never write to SSCBR, while the SSC is enabled.

The formulas below calculate the resulting Baud rate for a given reload value and the required reload value for a given Baud rate:

$$\text{Baud rate}_{SSC} = \frac{f_{CPU}}{2 \times [(SSCBR) + 1]}$$

$$SSCBR = \left(\frac{f_{CPU}}{2 \times \text{Baud rate}_{SSC}} \right) - 1$$

(SSCBR) represents the content of the reload register, taken as unsigned 16 bit integer.

Refer to the device datasheet for a table of Baud rates, reload values and resulting bit times.

SSCBR (F0B4h / 5Ah) ESFR Reset Value: 0000h

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

12.4 Error detection mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes, while Transmit Error and Baud rate Error only apply to slave mode. When an error is detected, the respective error flag is set. When the corresponding Error Enable bit is set, also an error interrupt request will be generated by setting SSCEIR (see *Figure 116*). 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 116).

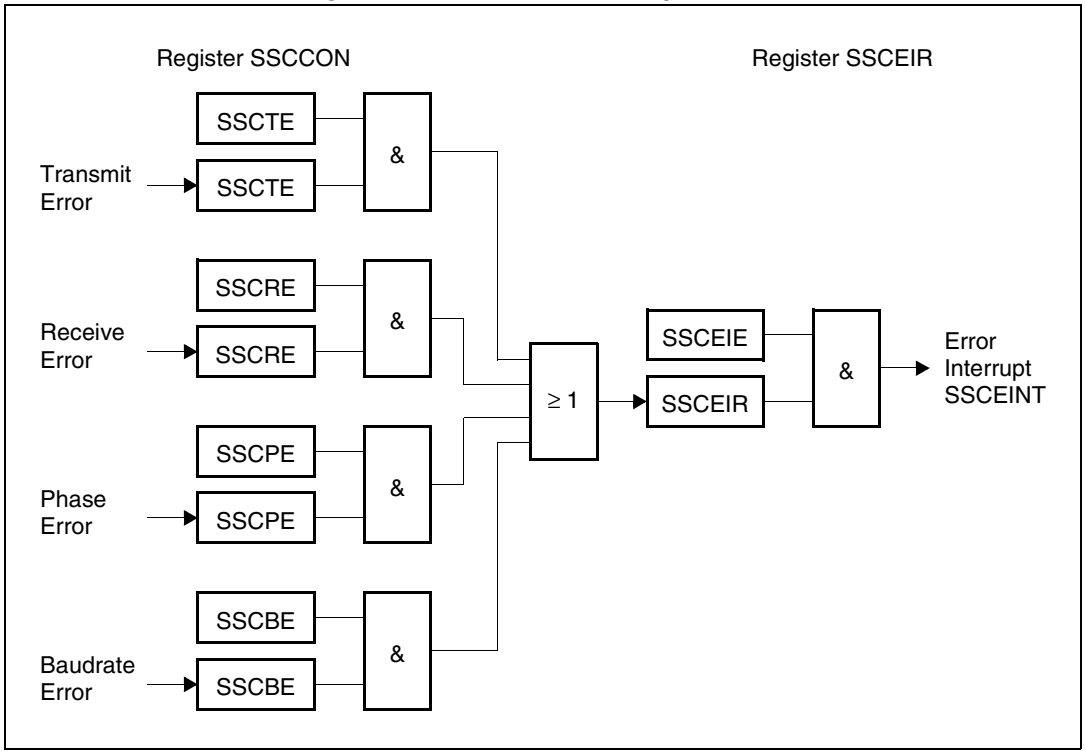
12.5 SSC interrupt control

Three interrupt control registers are provided for serial channel SSC. Register SSCTIC controls the transmit interrupt, SSCRIC controls the receive interrupt and SSCEIC controls the error interrupt of serial channel SSC. Each interrupt source also has its own dedicated interrupt vector. SCTINT is the transmit interrupt vector, SCRINT is the receive interrupt vector, and SCEINT is the error interrupt vector.

The cause of an error interrupt request (receive, phase, Baud rate, transmit error) can be identified by the error status flags in control register SSCCON.

Note: In contrary to the error interrupt request flag SSCEIR, the error status flags SSCxE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

Figure 116. SSC error interrupt control



SSCTIC (FF72h / B9h) SFR Reset Value: - - 00h

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

SSCRIC (FF74h / BAh) SFR Reset Value: - - 00h

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

SSCEIC (FF76h / BBh) SFR Reset Value: - - 00h

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

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

13 XBUS high-speed synchronous serial interface

A second High-Speed Synchronous Serial Interface (XSSC) is implemented on ST10F276. It is mapped on XBUS interface (Address range 00'E800h-00'E8FFh) and provides flexible high-speed serial communication between the ST10F276 and other microcontrollers, microprocessors or external peripherals. The XSSC is enabled by setting XPEN bit 2 of SYSCON register and bit 8 of XPERCON register.

The XSSC supports full-duplex and half-duplex synchronous communication. The serial clock signal can be generated by the XSSC 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 XSSC 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/XASC in synchronous mode), or configured like master / slave or multimaster interconnections or operate like the popular SPI interface. It can communicate with shift registers (I/O expansion), peripherals (such as EEPROMs) or other controllers (networking). The XSSC supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR1/P6.6 (Master Transmit / Slave Receive) and MRST1/P6.7 (Master Receive / Slave Transmit). The clock signal is output or input on pin SCLK1/P6.5. These pins are alternate functions of Port6 pins.

The main differences between SSC and XSSC are restricted to the programming model and interrupt management, due to the constraints imposed by the XBUS with respect to the standard ST10 peripheral bus (registers are not bit addressable, XBUS interrupt channels sharing with other X-Peripherals). In terms of general functionality and performance, the two modules are completely equivalent.

Figure 117. XBUS registers and port pins associated with the XSSC

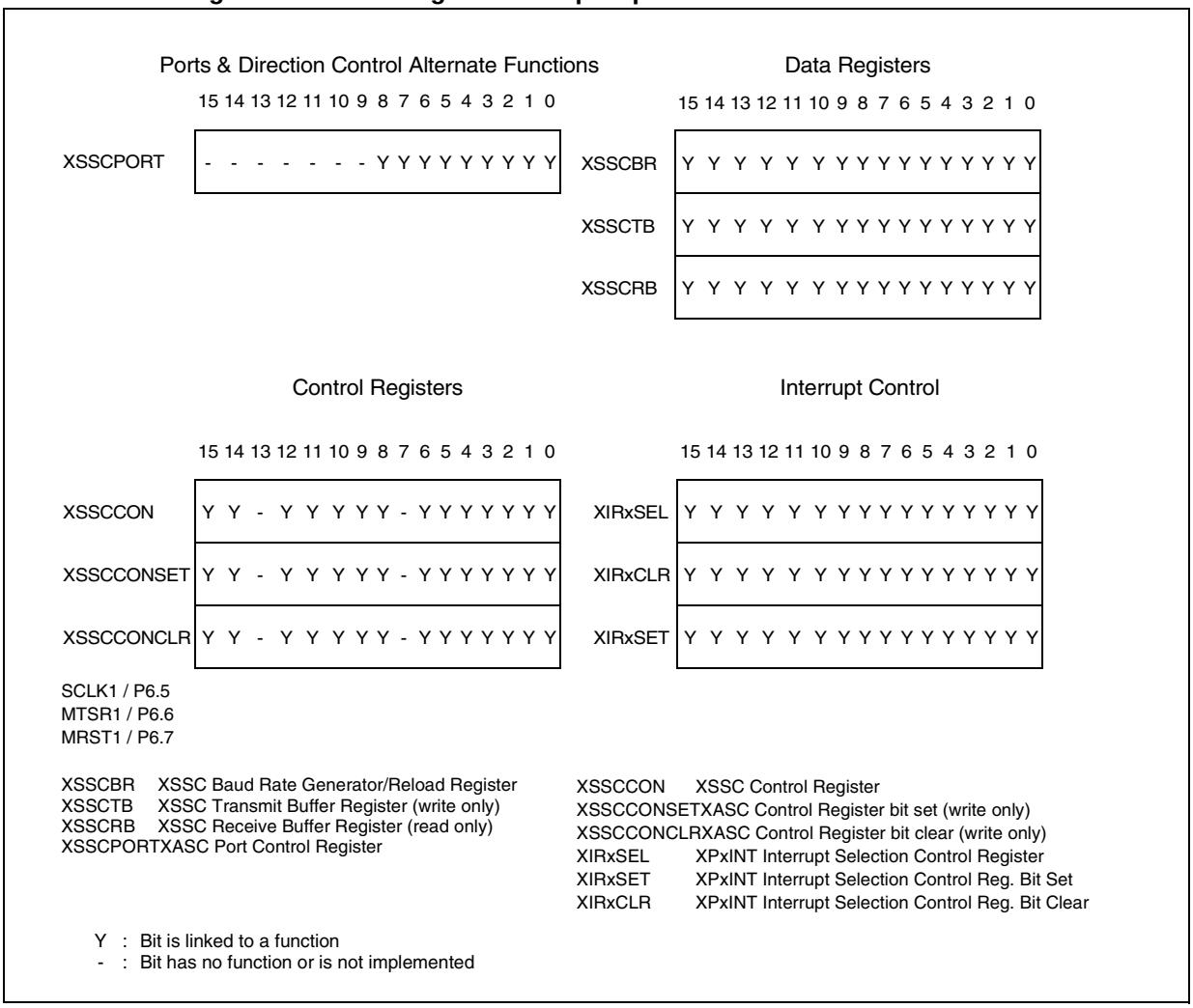
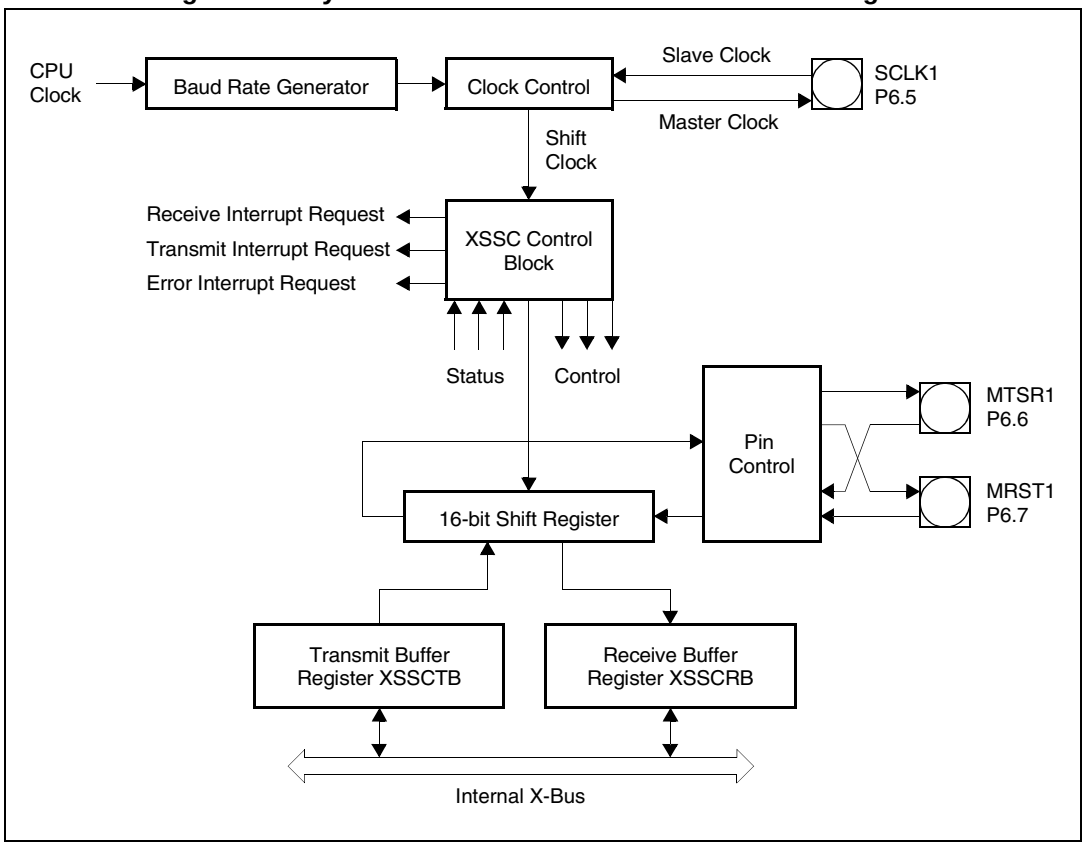


Figure 118. Synchronous serial channel XSSC block diagram



The operating mode of the serial channel XSSC is controlled by its bit-addressable control register XSSCCON. This register serves for two purposes:

- During programming (XSSC disabled by SSCEN = '0') it provides access to a set of control bits.
- During operation (XSSC enabled by SSCEN = '1') it provides access to a set of status flags. Register XSSCCON is shown below in each of the two modes.

XSSCCON (E800h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN=0	SSC MS	-	SSC AREN	SSC BEN	SSC PEN	SSC REN	SSC TEN	-	SSC PO	SSC PH	SSC HB	SSCBM			
RW	RW		RW	RW	RW	RW	RW		RW	RW	RW	RW			

Bit	Function (programming mode, SSCEN = '0')
SSCBM	**XSSC Data Width Selection** 0: Reserved. Do not use this combination. 1...15: Transfer Data Width is 2...16-bit [(SSCBM)+1]
SSCHB	**XSSC Heading Control bit** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First

Bit	Function (programming mode, SSCEN = '0')
SSCPH	**XSSC Clock Phase Control bit** 0: Shift transmit data on the leading clock edge, latch on trailing edge 1: Latch receive data on leading clock edge, shift on trailing edge
SSCPO	**XSSC Clock Polarity Control bit** 0: Idle clock line is low, leading clock edge is low-to-high transition 1: Idle clock line is high, leading clock edge is high-to-low transition
SSCTEN	**XSSC Transmit Error Enable bit** 0: Ignore transmit errors 1: Check transmit errors
SSCREN	**XSSC Receive Error Enable bit** 0: Ignore receive errors 1: Check receive errors
SSCPEN	**XSSC Phase Error Enable bit** 0: Ignore phase errors 1: Check phase errors
SSCBEN	**XSSC Baudrate Error Enable bit** 0: Ignore baudrate errors 1: Check baudrate errors
SSCAREN	**XSSC Automatic Reset Enable bit** 0: No additional action upon a baudrate error 1: The XSSC is automatically reset upon a baudrate error
SSCMS	**XSSC Master Select bit** 0: Slave Mode. Operate on shift clock received via SCLK1. 1: Master Mode. Generate shift clock and output it via SCLK1.
SSCEN	**XSSC Enable bit = '0'** Transmission and reception disabled. Access to control bits.

XSSCCON (E800h) XBUS 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					RW			

Bit	Function (operating mode, SSCEN = '1')
SSCBC	**XSSC bit Count Field** Shift counter is updated with every shifted bit. **Do not write to**
SSCTE	**XSSC Transmit Error Flag** 1: Transfer starts with the slave's transmit buffer not being updated
SSCRE	**XSSC Receive Error Flag** 1: Reception completed before the receive buffer was read
SSCPE	**XSSC Phase Error Flag** 1: Received data changes around sampling clock edge

Bit	Function (operating mode, SSCEN = '1')
SSCBE	**XSSC Baud rate Error Flag** 1: More than factor 2 or 0.5 between Slave's actual and expected Baud rate
SSCBSY	**XSSC Busy Flag:** Set while a transfer is in progress. **Do not write to**
SSCMS	**XSSC Master Select bit** 0: Slave Mode. Operate on shift clock received via SCLK1. 1: Master Mode. Generate shift clock and output it via SCLK1.
SSCEN	**XSSC Enable bit = '1'** Transmission and reception enabled. Access to status flags and M/S control.

Note: The target of an access to XSSCCON (control bits or flags) is determined by the state of SSCEN prior to the access. Writing C057h to XSSCCON in programming mode (SSCEN = '0') will initialize the XSSC (SSCEN was '0') and then turn it on (SSCEN = '1'). When writing to XSSCCON, make sure that reserved locations receive zeros.

All XSSCCON bits can be individually (bit-wise) programmed. The "bit-addressable" feature is available via specific "Set" and "Clear" registers: XSSCCONSET, XSSCCONCLR.

XSSCCONSET (E802h) XBUS Reset Value: 0000h

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

Bit	Function
SET.Y	**XSSCCON Bit Y Set** Writing a '1' will set the corresponding bit in XSSCCON register. Writing a '0' has no effect.

XSSCCONCLR (E804h) XBUS Reset Value: 0000h

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

Bit	Function
SET.Y	**XSSCCON Bit Y Clear** Writing a '1' will clear the corresponding bit in XSSCCON register. Writing a '0' has no effect.

The shift register of the XSSC is connected to both the transmit pin and the receive pin via the pin control logic (see *Figure 118*). 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 XSSCTB.

XSSCTB (E806h) XBUS Reset Value: 0000h

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

RW

XSSCRB (E808h) XBUS Reset Value: xxxxh

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

RW

It is moved to the shift register as soon as this is empty. An XSSC-master (SSCMS = ‘1’) immediately begins transmitting, while an XSSC-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 will be generated to indicate that XSSCTB 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 XSSCRB and a receive interrupt request will be generated. If no further transfer is to take place (XSSCTB is empty), SSCBSY will be cleared at the same time. Software should not modify SSCBSY, as this flag is hardware controlled. Only one XSSC can be master at a given time.

The transfer of serial data bit can be programmed in the following ways:

- The data width can be chosen from 2 bits to 16 bits.
- Transfer may start with the LSB or the MSB.
- The shift clock may be idle low or idle high.
- Data bit may be shifted with the leading or trailing edge of the clock signal.
- The Baud rate may be set for a range of values (refer to *Section 12.3: Baud rate generation on page 280* 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 XSSC 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 XSSCTB and XSSCRB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of XSSCTB are ignored, the unselected bits of XSSCRB 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 XSSC 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. *Figure 113 on page 276* is a summary.

13.1 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 MTSR1 is the transmit line, the receive line is connected to its data input line MRST1, and the clock line is connected to pin SCLK1. Only the device selected for master operation generates and outputs the serial clock on pin SCLK1. All slaves receive this clock, so their pin SCLK1 must be switched to input mode (XDP6.5 = '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 Figure 113 on page 276 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 XSSC and also the function of the respective port lines (see *Section 12.2.1: Port control on page 279*).

Figure 119. Serial clock phase and polarity options

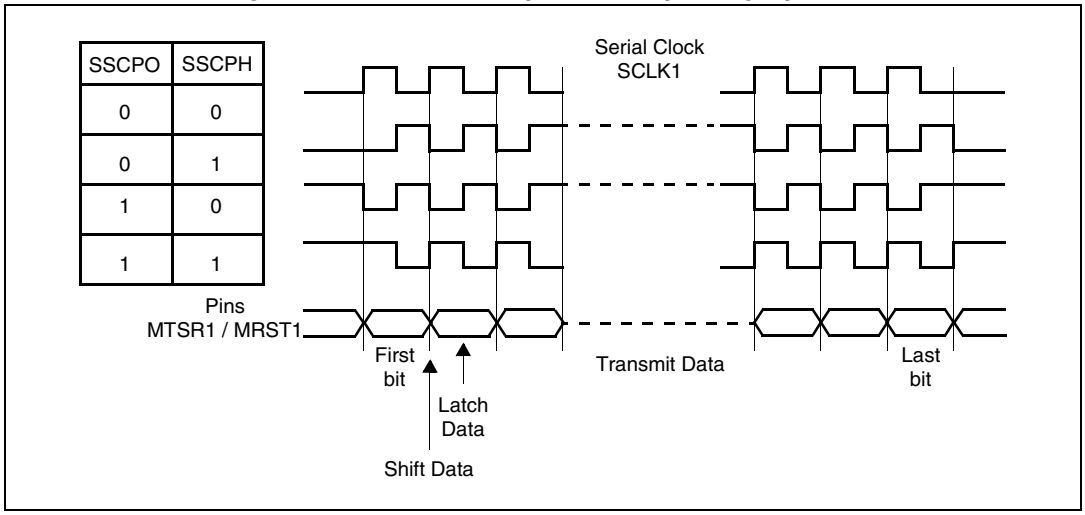
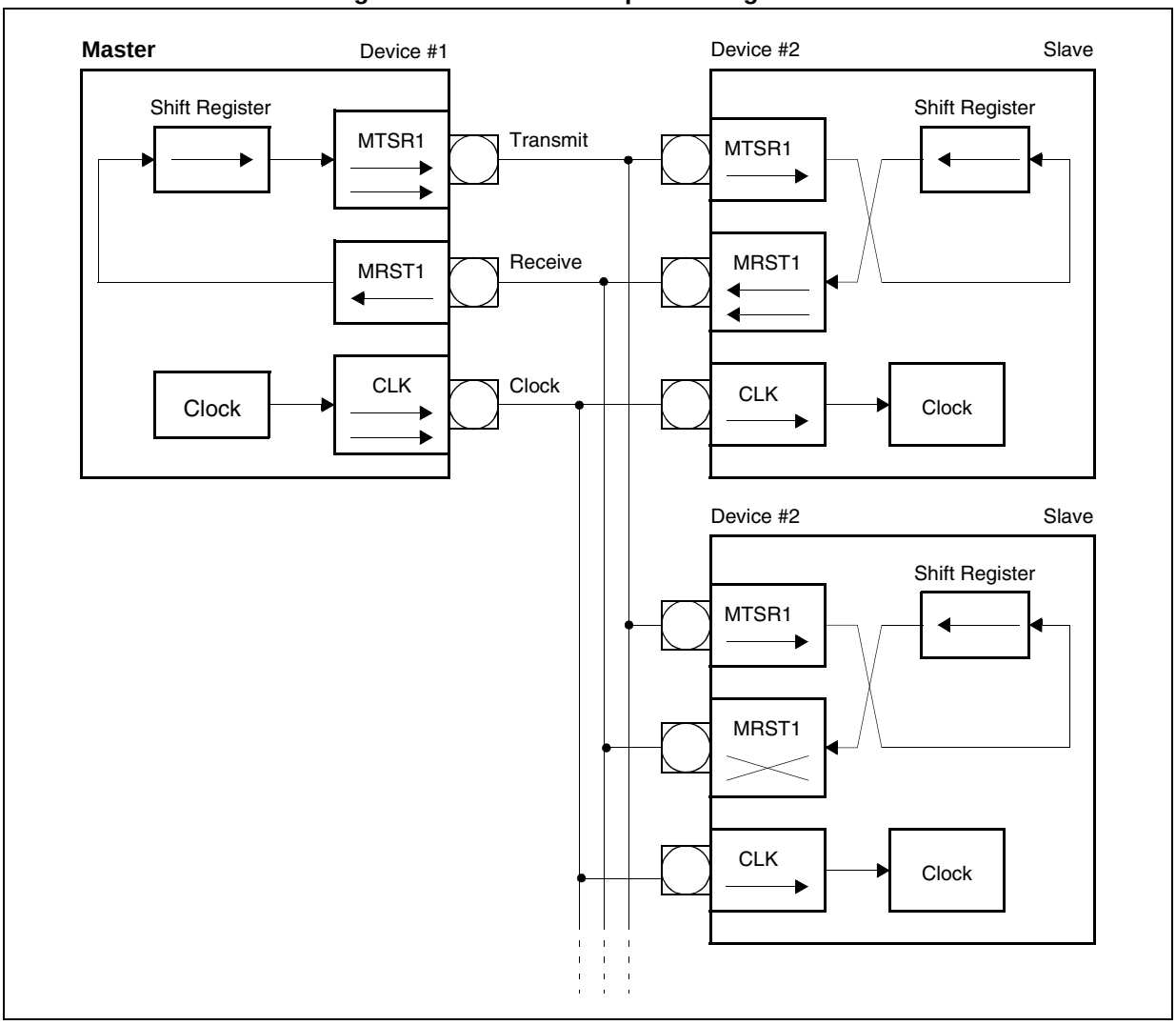


Figure 120. XSSC full duplex configuration



The data output pins MRST1 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 MRST1 pin. All the other slaves have to program there MRST1 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 MRST1 line to output, until it gets a de-selection signal or command.

The slaves use open drain output on MRST1. This forms a AND-wired connection. The receive line needs an external pull-up in this case. Corruption of the data on the receive line sent by the selected slave is avoided, when all slaves which are not selected for transmission to the master only send ones (‘1’). Since this high level is not actively driven onto the line, but only held through the pull-up device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from

which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initialization of the XSSC, 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 XSSCTB. 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 MTSR1 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 SCLK1 line.

With the opposite clock edge the master at the same time latches and shifts in the data detected at its input line MRST1. 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 XSSCRB 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 MRST1, 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 SCLK1 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 XSSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the XSSC with an idle-low clock (SSCPO = ‘0’) will drive the alternate data output and (via the AND) the port pin SCLK1 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 (XP6.5 = ‘x’)
- Switch the pin to output (XDP6.5 = ‘1’)
- Enable the XSSC (SSCEN = ‘1’)
- If SSCPO = ‘0’: enable alternate data output (XP6.5 = ‘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).

13.2 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 MTSR1 and MRST1 of each device, the clock line is connected to the SCLK1 pin.

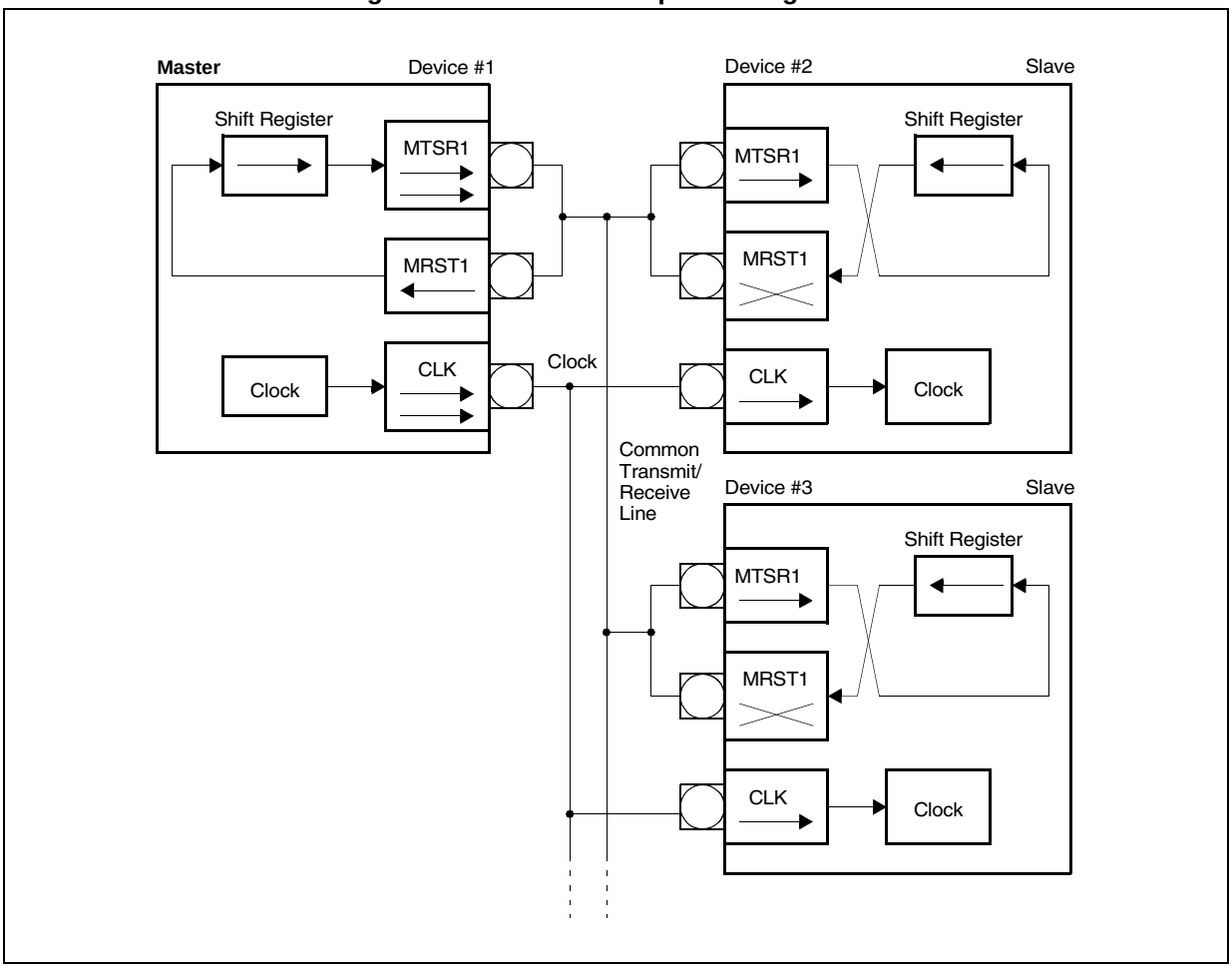
The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

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 (MRST1 for a master device, MTSR1 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 121. XSSC half duplex configuration



Continuous transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer XSSCTB is empty and ready to be loaded with the next transmit data. If XSSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line there is no gap between the two successive frames, so two bytes transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. It is just a matter of software, how long a total data frame length can be. This option can also be used to interface to byte wide and word wide devices on the same serial bus.

Note: Of course, this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the XSSC to reprogram the basic data width on-the-fly.

13.2.1 Port control

The XSSC uses three pins of Port6 to communicate with the external world. Pin P6.5/SCLK1 serves as the clock line, while pins P6.7/MRST1 (Master Receive / Slave Transmit) and P6.6/MTSR1 (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 XSSC 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 next table (the table reports the indication of bit names inside XSSCPORT register). Using the open drain output feature helps to avoid bus contention problems and reduces the need for hard-wired 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
P6.5 / SCLK1	Serial Clock Output	XP6.5 = ‘1’	XDP6.5 = ‘1’	Serial Clock Input	XP6.5 = ‘x’	XDP6.5 = ‘0’
P6.6 / MTSR1	Serial Data Output	XP6.6 = ‘1’	XDP6.6 = ‘1’	Serial Data Input	XP6.6 = ‘x’	XDP6.6 = ‘0’
P6.7 / MRST1	Serial Data Input	XP6.7 = ‘x’	XDP6.7 = ‘0’	Serial Data Output	XP6.7 = ‘1’	XDP6.7 = ‘1’

Note: In the table above, an 'x' means that the actual value is irrelevant in the respective mode, however, it is recommended to set these bits to '1', so they are already in the correct state when switching between master and slave mode.

13.3 Baud rate generation

The serial channel XSSC 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 XSSCCON. Register XSSCBR is the dual-function Baud Rate Generator/Reload register. Reading XSSCBR, while the XSSC is enabled, returns the content of the timer. Reading XSSCBR, while the XSSC is disabled, returns the programmed reload value. In this mode the desired reload value can be written to XSSCBR.

Note: Never write to XSSCBR, while the XSSC 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:

$$\text{Baud rate}_{XSSC} = \frac{f_{CPU}}{2 \times [(XSSCBR) + 1]}$$

$$XSSCBR = \left(\frac{f_{CPU}}{2 \times \text{Baud rate}_{XSSC}}\right) - 1$$

(XSSCBR) 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.

XSSCBR (E80Ah) XBUS Reset Value: 0000h

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

13.4 Error detection mechanisms

The XSSC 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. 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, 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 XSSCRB. This condition sets the error flag SSCRE and, when enabled via SSCREN, the error interrupt request flag (see XP3INT line). The old data in the receive buffer XSSCRB 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 MRST1 (master mode) or MTSR1 (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 (see XP3INT line).

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 (see XP3INT line). 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 XSSC will be performed in case of this error. This is done to re-initialize the XSSC, 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 XSSCTB 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 (see XP3INT line). 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.

13.5 XSSC interrupt control

Up to four interrupt control registers (XIRxSEL, x = 0, 1, 2, 3) are provided in order to select the source of the XBUS interrupt: the transmit interrupt, the receive interrupt and the error interrupt of serial channel XSSC are linked to the one of the XPxIC registers (x = 0, 1, 2, 3). In particular, the three interrupt lines are available on the following interrupt vectors:

- Receive XP0INT XP1INT XP2INT
- Transmit XP0INT XP1INT XP2INT
- Error XP3INT

Refer to *Section 5.7: X-peripheral interrupt on page 117* for details.

The cause of an error interrupt request (receive, phase, baudrate, transmit error) can be identified by checking the error status flags in control register XSSCCON.

Note: The error status flags SSCxE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

14 Watchdog timer

The watchdog timer (WDT) provides recovery from software or hardware failure. If the software fails to service this timer before an overflow occurs, an internal reset sequence is initiated.

This internal reset will also pull the $\overline{\text{RSTOUT}}$ pin low, this resets the peripheral hardware which might have caused the malfunction. When the watchdog timer is enabled and is serviced regularly to prevent overflows, the watchdog timer supervises program execution. Overflow only occurs if the program does not progress properly.

The watchdog timer will time out, if a software error was due to hardware related failures. This prevents the controller from malfunctioning for longer than a user-specified time.

The watchdog timer provides two registers:

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

The watchdog timer is a 16-bit up counter which can be clocked with the CPU clock (f_{CPU}) either divided by 2 or divided by 128. This 16-bit timer is realized as two concatenated 8-bit timers (see *Figure 123*).

The upper 8 bits of the watchdog timer can be preset to a user-programmable value by a watchdog service access, in order to program the watchdog expire time. The lower 8 bits are reset on each service access.

Figure 122. SFRs and port pins associated with the watchdog timer

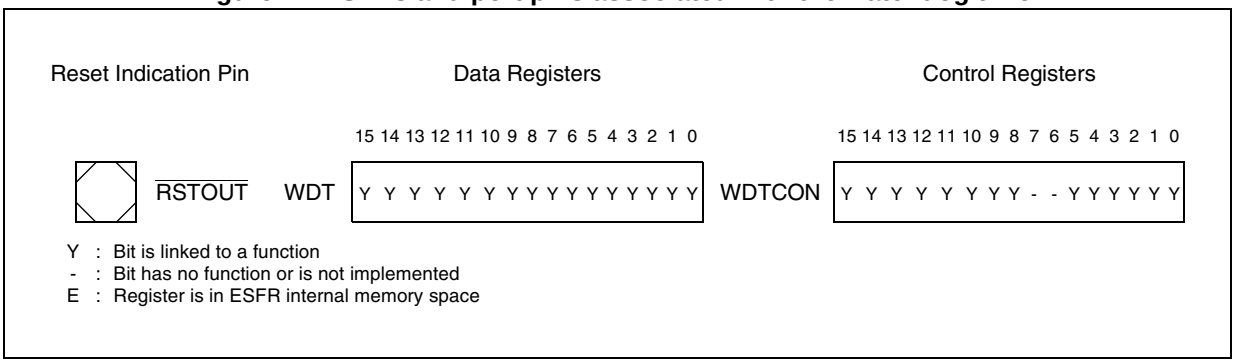
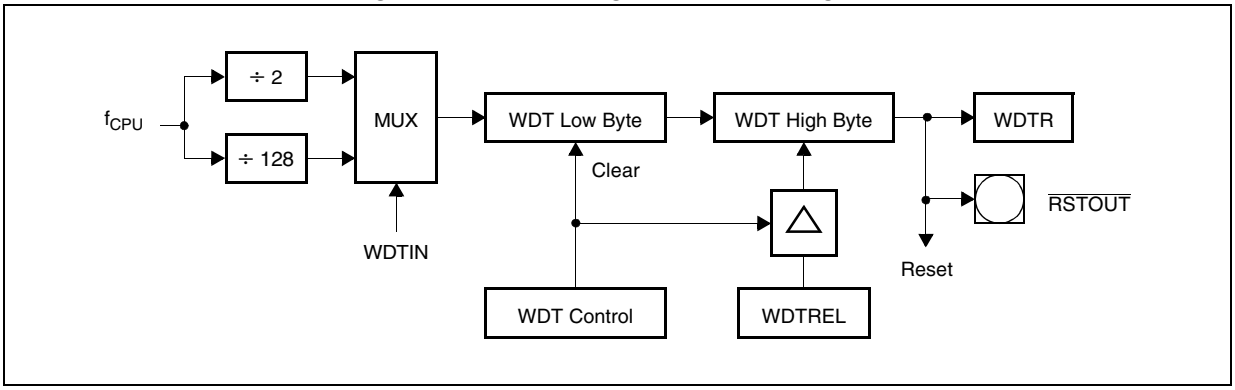


Figure 123. Watchdog timer block diagram



14.1 Operation of the watchdog timer

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

WDTCON (FFAEh / D7h) SFR Reset Value: 00xxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								-	-	**PONR**	**LHWR**	**SHWR**	**SWR**	**WDTR**	WDTIN
RW										RW	RW	RW	RW	RW	RW

WDTIN	**Watchdog Timer Input Frequency Selection** '0': Input Frequency is $f_{CPU}/2$ (Default after Reset). '1': Input Frequency is $f_{CPU}/128$.
WDTR(1)	**Watchdog Timer Reset Indication Flag** Set by the watchdog timer on an overflow. Cleared by a hardware reset or by the SRVWDT instruction.
SWR(1)	**Software Reset Indication Flag** Set by the SRST execution. Cleared by the EINIT instruction.
SHWR(1)	**Short Hardware Reset Indication Flag** Set by the input $\overline{\text{RSTIN}}$. Cleared by the EINIT instruction.
LHWR(1)	**Long Hardware Reset Indication Flag** Set by the input $\overline{\text{RSTIN}}$. Cleared by the EINIT instruction.
PONR(1) - (2)	**Power-On (Asynchronous) Reset Indication Flag** Set by the input $\overline{\text{RSTIN}}$ if a Power-On condition has been detected. Cleared by the EINIT instruction.

1. More than one reset indication flag may be set. After EINIT, all flags are cleared.
2. Power-on is detected when a rising edge from V_{DD} = 0 V to V_{DD} > 2.0 V is recognized.

After any software reset, external hardware reset (see note), or watchdog timer reset, the watchdog timer is enabled and starts counting up from 0000h with the frequency f_{CPU} / 2. The input frequency may be switched to f_{CPU} / 128 by setting bit WDTIN. The watchdog timer can be disabled via the instruction DISWDT (Disable Watchdog Timer). Instruction DISWDT is a protected 32-bit instruction which will ONLY be executed during the time between a reset and execution of either the EINIT (End of Initialization) or the SRVWDT (Service Watchdog Timer) instruction. Either one of these instructions disables the execution of DISWDT.

When the watchdog timer is not disabled via instruction DISWDT, it will continue counting up, even during Idle Mode. If it is not serviced via the instruction SRVWDT by the time the count reaches FFFFh the watchdog timer will overflow and cause an internal reset. This reset will pull the external reset indication pin $\overline{\text{RSTOUT}}$ low. It differs from a software or external hardware reset in that bit WDTR (Watchdog Timer Reset Indication Flag) of register WDTCON will be set. A hardware reset or the SRVWDT instruction will clear this bit. Bit WDTR can be examined by software in order to determine the cause of the reset.

A watchdog reset will also complete a running external bus cycle before starting the internal reset sequence if this bus cycle does not use READY or samples READY active (low) after the programmed wait-states. Otherwise the external bus cycle will be aborted.

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

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

After being serviced the watchdog timer continues counting up from the value [(WDTREL) x 2^8]. Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer (for example, by fetching and executing a bit pattern from a wrong location) is minimized. When instruction SRVWDT does not match the format for protected instructions, the Protection Fault Trap will be entered, rather than the instruction be executed.

The PONR flag of WDTCON register is set if the output voltage of the internal 1.8V supply falls below the threshold (typically 1.65V) of the Power-On detection circuit. This circuit is efficient to detect major failures of the external 5V supply but if the internal 1.8V supply does not drop under 1.65 volts, the PONR flag is not set. This could be the case on fast switch-off / switch-on of the 5V supply. The time needed for such a sequence to activate the PONR flag depends on the value of the capacitors connected to the supply and on the exact value of the internal threshold of the detection circuit.

Table 42. WDTCON bits value on different resets

Reset source	PONR	LHWR	SHWR	SWR	WDTR
Power-On Reset	X	X	X	X	
Power-On after partial supply failure	Note 1	X	X	X	
Long Hardware Reset		X	X	X	
Short Hardware Reset			X	X	
Software Reset				X	
Watchdog Reset				X	X

Note: 1 PONR bit may not be set for short supply failure.

2 For Power-On reset and reset after supply partial failure, asynchronous reset must be used.

In *Table 44* a summary of the different reset events and consequent WDTCON flag setting is reported. Refer also to *Section 23: System reset on page 472* for details.

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

Each time it is serviced by the application software, the high byte of the watchdog timer is reloaded. For security, rewrite WDTCON each time before the watchdog timer is serviced.

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

- **The input frequency** to the watchdog timer can be selected via bit WDTIN in register WDTCON to be either f_{CPU} / 2 or f_{CPU} / 128.
- **The reload value** WDTREL for the high byte of WDT can be programmed in register WDTCON.

The period P_{WDT} between servicing the watchdog timer and the next overflow can therefore be determined by the following formula:

$$P_{WDT} = \frac{2^{[1 + (WDTIN) \times 6]} \times [2^{16} - (WDTREL) \times 2^{8}]}{f_{CPU}}$$

Refer to the device datasheet for a table of watchdog timer ranges. For security, you are advised to rewrite WDTCON each time before the watchdog timer is serviced.

The *Table 43* shows the watchdog time range for 40 MHz and 64 MHz CPU clock.

Table 43. WDTREL reload value

Reload value in WDTREL	Prescaler for f_{CPU} = 40 MHz		Prescaler for f_{CPU} = 64 MHz	
	2 (WDTIN = '0')	128 (WDTIN = '1')	2 (WDTIN = '0')	128 (WDTIN = '1')
FFh	12.8µs	819.2µs	8µs	512µs
00h	3.277ms	209.7ms	2.048ms	131.1ms

Table 44. Reset events summary

Event	RPD	EA	Bidir	Synch. Asynch.	RSTIN		WDTCON Flags				
					min	max	PONR	LHWR	SHWR	SWR	WDTR
Power-on Reset	0	0	N	Asynch.	1 ms (VREG) 1.2 ms (Reson. + PLL) 10.2 ms (Crystal + PLL)	-	1	1	1	1	0
	0	1	N	Asynch.	1ms (VREG)	-	1	1	1	1	0
	1	x	x	Forbidden							
	x	x	Y	Not applicable							
Hardware reset (Asynchronous)	0	0	N	Asynch.	500ns	-	0	1	1	1	0
	0	1	N	Asynch.	500ns	-	0	1	1	1	0
	0	0	Y	Asynch.	500ns	-	0	1	1	1	0
	0	1	Y	Asynch.	500ns	-	0	1	1	1	0

Table 44. Reset events summary (continued)

Event	RPD	EA	Bidir	Synch. Asynch.	RSTIN		WDTCON Flags				
					min	max	PONR	LHWR	SHWR	SWR	WDTR
Short hardware reset (Synchronous) (1)	1	0	N	Synch.	max (4TCL, 500ns)	1032TCL	0	0	1	1	0
	1	1	N	Synch.	max (4TCL, 500ns)	1032TCL	0	0	1	1	0
	1	0	Y	Synch.	max (4TCL, 500ns)	1032TCL	0	0	1	1	0
					Activated by internal logic for 1024TCL						
	1	1	Y	Synch.	max (4TCL, 500ns)	1032TCL	0	0	1	1	0
					Activated by internal logic for 1024TCL						
Long Hardware Reset (Synchronous)	1	0	N	Synch.	1032TCL	-	0	1	1	1	0
	1	1	N	Synch.	1032TCL	-	0	1	1	1	0
	1	0	Y	Synch.	1032TCL	-	0	1	1	1	0
					Activated by internal logic only for 1024TCL						
	1	1	Y	Synch.	1032TCL	-	0	1	1	1	0
					Activated by internal logic only for 1024TCL						
Software Reset (2)	x	0	N	Synch.	Not activated		0	0	0	1	0
	x	0	N	Synch.	Not activated		0	0	0	1	0
	0	1	Y	Synch.	Not activated		0	0	0	1	0
	1	1	Y	Synch.	Activated by internal logic for 1024TCL		0	0	0	1	0
Watchdog Reset (2)	x	0	N	Synch.	Not activated		0	0	0	1	1
	x	0	N	Synch.	Not activated		0	0	0	1	1
	0	1	Y	Synch.	Not activated		0	0	0	1	1
	1	1	Y	Synch.	Activated by internal logic for 1024TCL		0	0	0	1	1

1. It can degenerate into a long hardware reset and consequently differently flagged (see *Section 24.3* for details).
2. When bidirectional is active (and with RPD = 0), it can be followed by a short hardware reset and consequently differently flagged (see *Section 23.6: Bidirectional reset* for details).

15 The bootstrap loader

ST10F276 implements innovative Boot capabilities in order to:

- Support a user defined bootstrap (see ‘Alternate Bootstrap Loader’);
- Support bootstrap via UART or bootstrap via CAN for the standard bootstrap.

15.1 Selection among user-code, standard or alternate bootstrap

The selection among user-code, standard bootstrap or alternate bootstrap is made by special combinations on Port0L[5...4] during the time the reset configuration is latched from Port0.

The alternate boot mode is triggered with a special combination set on Port0L[5...4]. Those signals, as other configuration signals are latched on the rising edge of RSTIN pin.

The alternate boot function is divided in two functional parts (which are independent from each other):

Part 1: Selection of reset sequence according to the Port0 configuration, the User Mode and the Alternate Mode signatures

- Decoding of reset configuration (P0L.5 = 1, P0L.4 = 1) will select the normal mode and select the user Flash to be mapped from address 00’0000h.
- Decoding of reset configuration (P0L.5 = 1, P0L.4 = 0) will select ST10 standard bootstrap mode (Test-Flash is active and overlaps user Flash for code fetches from address 00'0000h; user Flash is active and available for read and program).
- Decoding of reset configuration (P0L.5 = 0, P0L.4 = 1) will activate new verifications to select which bootstrap software to execute:
- if the User mode signature in the User Flash is programmed correctly, then a software reset sequence is selected and the User code is executed;
- if the User mode signature is not programmed correctly but the Alternate mode signature in the User Flash is programmed correctly, then the Alternate Boot Mode is selected;
- if both the User and the Alternate mode signatures are not programmed correctly in the user Flash, then the User key location is read again. Its value will determine the behavior of the Selective Bootstrap Loader.

Part 2: Running of user selected reset sequence

- Standard Bootstrap Loader: jump to a predefined memory location in Test-Flash (controlled by ST)
- Alternate Boot Mode: jump to address 09’0000h
- Selective Bootstrap Loader: jump to a predefined location in Test-Flash (controlled by ST) and check which communication channel is selected
- User code: make a software reset and jump to 00’0000h

Table 45. ST10F276x boot mode selection

P0.5	P0.4	ST10 decoding
1	1	**User Mode:** user Flash mapped at 00'0000h
1	0	**Standard Bootstrap Loader:** User Flash mapped from 00'0000h, code fetches redirected to Test-Flash at 00'0000h
0	1	**Alternate Boot Mode:** Flash mapping depends on signatures integrity check
0	0	Reserved

15.2 Standard bootstrap loader

The built-in bootstrap loader of the ST10F276 provides a mechanism to load the startup program, which is executed after reset, via the serial interface. In this case no external (ROM) memory or an internal ROM is required for the initialization code starting at location $00'0000_H$. The bootstrap loader moves code/data into the IRAM, but it is also possible to transfer data via the serial interface into an external RAM using a second level loader routine. ROM memory (internal or external) is not necessary. However, it may be used to provide lookup tables or may provide "core-code", that is, a set of general purpose subroutines, for example, such as for I/O operations, number crunching, or system initialization.

The Bootstrap Loader may be used to load the complete application software into ROMless systems, it may load temporary software into complete systems for testing or calibration, it may also be used to load a programming routine for Flash devices.

The BSL mechanism may be used for standard system startup as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

15.2.1 Entering the standard bootstrap loader

As with the old ST10 bootstrap mode, the ST10F276 enters BSL mode, if pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in bootstrap loader is activated independently of the selected bus mode. The bootstrap loader code is stored in a special Test-Flash: no part of the standard of the Flash memory area is required for this.

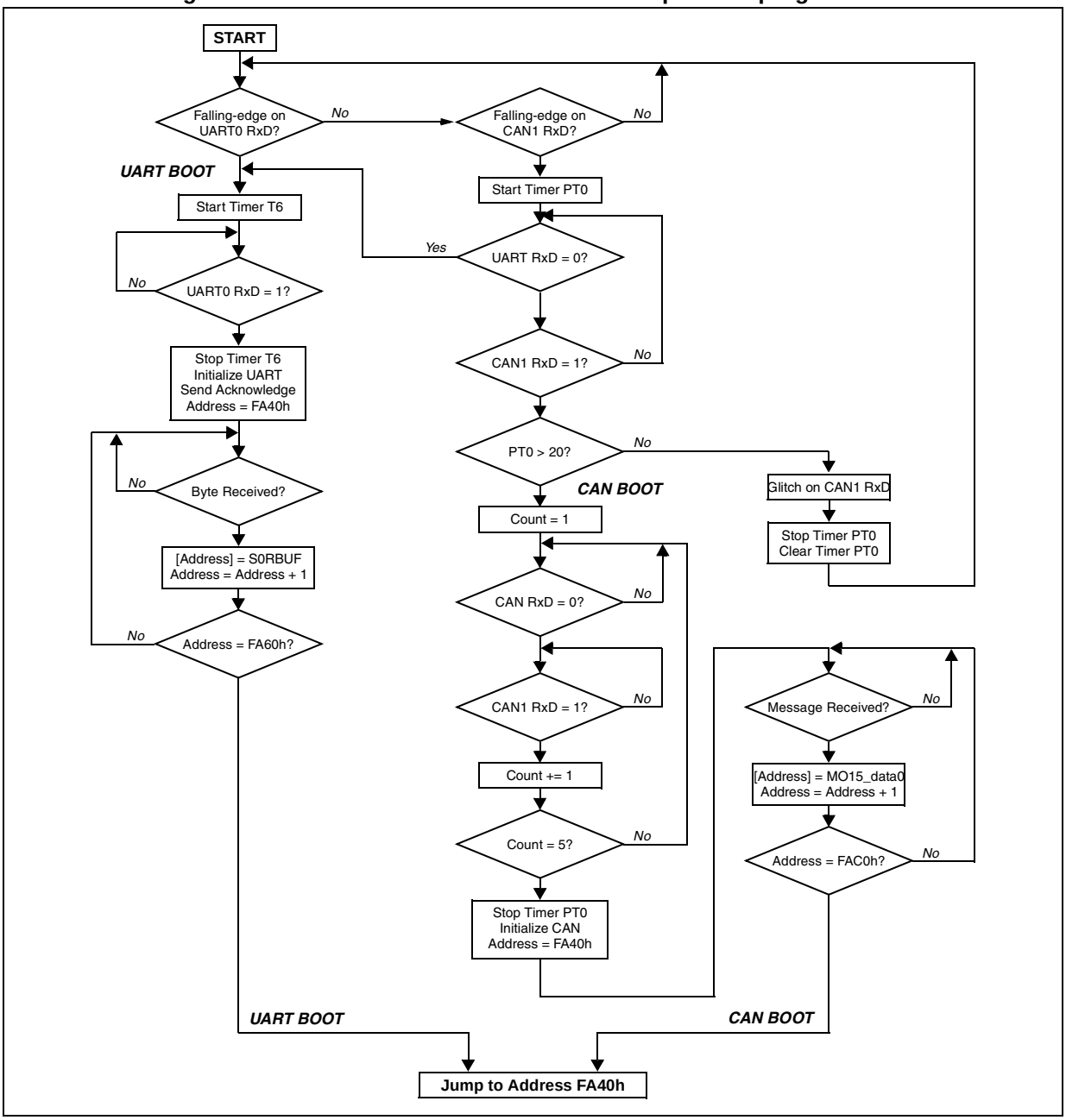
After entering BSL mode and the respective initialization, the ST10F276 scans the RxD0 line and the CAN1_RxD line to receive either a valid dominant bit from CAN interface, or a start condition from UART line.

Start condition on UART RxD: ST10F276 starts standard bootstrap loader. This bootstrap loader is identical to other ST10 devices (example: ST10F269, ST10F168). See *Section 15.3: Standard bootstrap with UART (RS232 or K-Line)* for details.

Valid dominant bit on CAN1 RxD: ST10F276 start bootstrapping via CAN1; the bootstrapping method is new and is described in the *Section 15.4: Standard bootstrap with CAN*. *Figure 124* shows the program flow of the new bootstrap loader. It illustrates clearly how the new functionalities are implemented:

- UART: UART has priority over CAN after a falling edge on CAN1_RxD till the 1st valid rising edge on CAN1_RxD;
- CAN: pulses on CAN1_RxD shorter than 20*CPU-cycles are filtered.

Figure 124. ST10F276x new standard bootstrap loader program flow



15.2.2 ST10 configuration in BSL

When the ST10F276 has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked in bold***):

Watchdog Timer	**Disabled**	
Register SYSCON:	***0404$_H$*** (1)	*XPEN bit set for Bootstrap via ; CAN or Alternate Boot Mode*
Context Pointer CP:	$FA00_H$	
Register STKUN:	$FC00_H$	
Stack Pointer SP:	***FA40$_H$***	
Register STKOV:	$FA00_H$	
Register BUSCON0:	acc. to startup config.(2)	
Register S0CON:	***8011$_H$***	*Initialized only if Bootstrap via UART*
Register S0BG:	acc. to '00' byte	*; Initialized only if Bootstrap via UART*
P3.10 / TXD0:	note (1)	*Initialized only if Bootstrap via UART*
DP3.10:	note (1)	*Initialized only if Bootstrap via UART*
CAN1 Status/Control register:	***0000$_H$***	*Initialized only if Bootstrap via CAN*
CAN1 Bit Timing Register:	acc. to '0' frame	*Initialized only if Bootstrap via CAN*
Register XPERCON:	***042D$_H$***	*XRAM1-2, XFlash, CAN1 and XMISC enabled Initialized only if Bootstrap via CAN*
P4.6 / CAN1_TxD:	note (1)	*Initialized only if Bootstrap via CAN*
DP4.6:	note (1)	*Initialized only if Bootstrap via CAN*

1. In Bootstrap modes (standard or alternate) ROMEN, bit 10 of SYSCON, is always set regardless of EA pin level. BYTDIS, bit 9 of SYSCON, is set according to data bus width selection via Port0 configuration.
2. BUSCON0 is initialized with 0000h, external bus disabled, if pin EA is high during reset. If pin EA is low during reset, BUSACT0, bit 10, and ALECTL0, bit 9, are set enabling the external bus with lengthened ALE signal. BTYP field, bit 7 and 6, is set according to Port0 configuration.

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Depending on the selected serial link (UART0 or CAN1), pin TxD0 or CAN1_TxD is configured as output, so the ST10F276 can return the acknowledge byte. Even if the internal IFLASH is enabled, no code can be executed out of it.

15.2.3 Booting steps

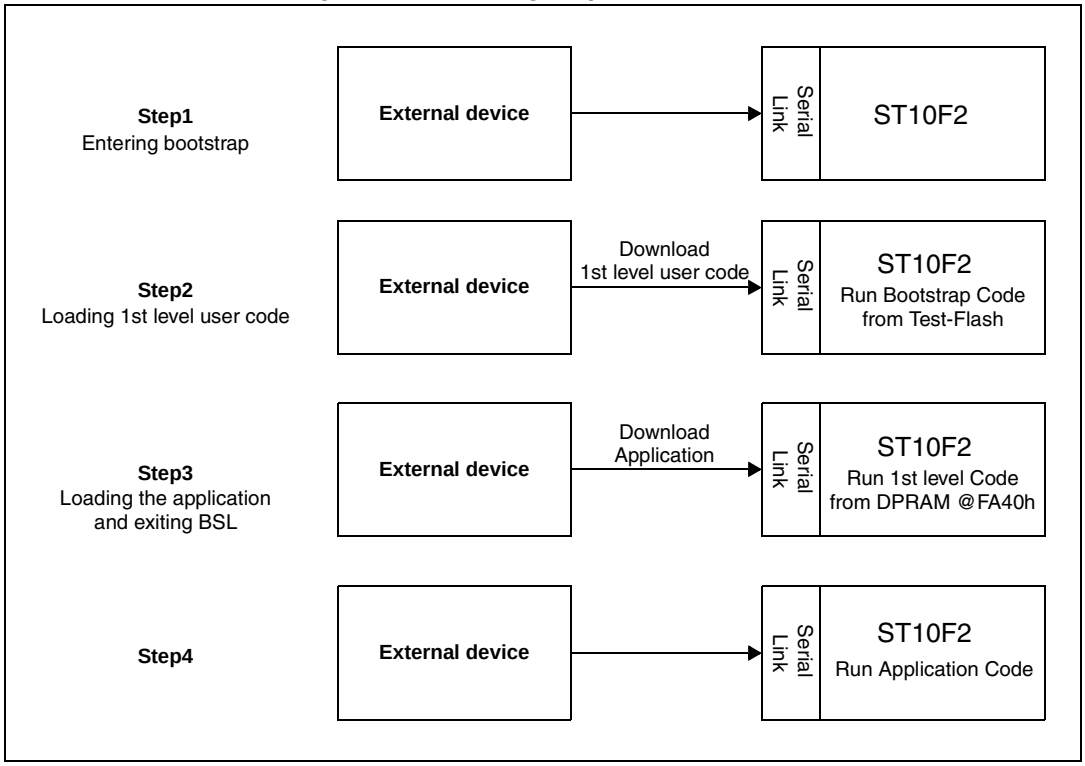
As *Figure 125* shows, booting ST10F276 with the boot loader code occurs in four steps minimum:

1. The ST10F276 is reset with P0L.4 low.
2. The internal new bootstrap code runs on the ST10 and a first level user code is downloaded from the external device, via the selected serial link (UART0 or CAN1). The bootstrap code is contained in the ST10F276 Test-Flash and is automatically run

when ST10F276 is reset with P0L.4 low. After loading a preselected number of bytes, ST10F276 begins executing the downloaded program.

3. The First level user code run on ST10F276. Typically, this first level user code is another loader that is used to download the application software into the ST10F276.
4. The loaded application software is now running.

Figure 125. Booting steps for ST10F276

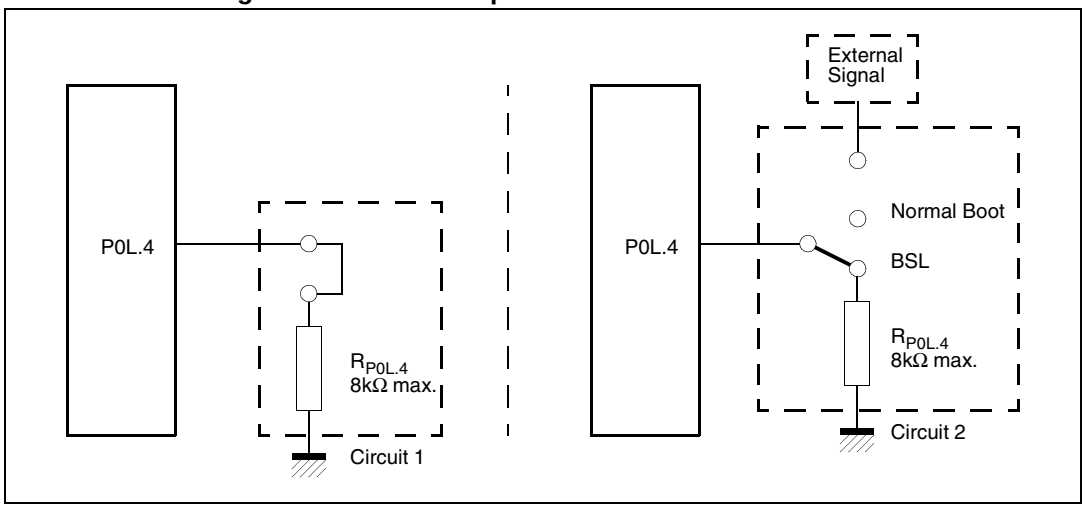


15.2.4 Hardware to activate BSL

The hardware that activates the BSL during reset may be a simple pull-down resistor on P0L.4 for systems that use this feature upon every hardware reset. You may want to use a switchable solution (via jumper or an external signal) for systems that only temporarily use the bootstrap loader.

Note: CAN alternate function on Port4 lines is not activated if the user has selected eight address segments (Port4 pins have three functions: I/O port, address-segment, CAN). Boot via CAN requires that four address segments or less are selected.

Figure 126. Hardware provisions to activate the BSL



15.2.5 Memory configuration in bootstrap loader mode

The configuration (that is, the accessibility) of the ST10F276's memory areas after reset in Bootstrap Loader mode differs from the standard case. Pin $\overline{EA}$ is evaluated when BSL mode is selected in order to enable or not the external bus:

- If $\overline{EA}$ = 1, the external bus is disabled (BUSACT0 = 0 in BUSCON0 register);
- If $\overline{EA}$ = 0, the external bus is enabled (BUSACT0 = 1 in BUSCON0 register).

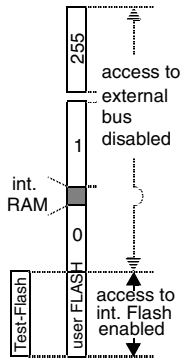
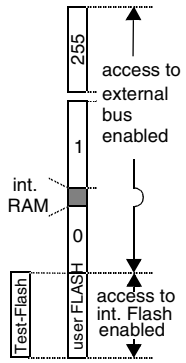
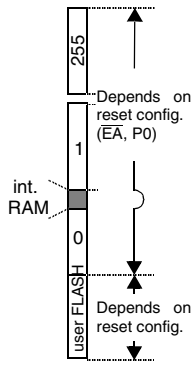
Moreover, while in BSL mode, accesses to the internal IFLASH area are partly redirected:

- All code accesses are made from the special Test-Flash seen in the range 00'0000h to 00'01FFFh;
- User IFLASH is only available for read and write accesses (Test-Flash cannot be read nor written);
- Write accesses must be made with addresses starting in segment 1 from 01'0000h, whatever the value of ROMS1 bit in SYSCON register;
- Read accesses are made in segment 0 or in segment 1 depending of ROMS1 value;
- In BSL mode, by default, ROMS1 = 0 so the first 32 Kbytes of IFlash are mapped in segment 0.

Example:

In default configuration, to program address 0, user must put the value 01'0000h in the FARL and FARH registers, but to verify the content of the address 0 a read to 00'0000h must be performed.

Figure 127. Memory configuration after reset

	16 Mbytes	16 Mbytes	16 Mbytes
			
BSL mode active	**Yes** (P0L.4 = ‘0’)	**Yes** (P0L.4 = ‘0’)	**No** (P0L.4 = ‘1’)
$\overline{EA}$ pin	high	low	according to application
Code fetch from internal FLASH area	Test-FLASH access	Test-FLASH access	User IFLASH access
Data fetch from internal FLASH area	User IFLASH access	User IFLASH access	User IFLASH access

Note: As long as ST10F276 is in BSL, user’s software should not try to execute code from the internal IFlash as the fetches are redirected to the Test-Flash.

15.2.6 Loading the start-up code

After the serial link initialization sequence (see following sections), the BSL enters a loop to receive 32bytes (boot via UART) or 128bytes (boot via CAN).

These bytes are stored sequentially into ST10F276 Dual-Port RAM from location 00’FA40h.

To execute the loaded code, the BSL then jumps to location 00’FA40h. The bootstrap sequence running from the Test-Flash is now terminated; the microcontroller remains in BSL mode however.

Most probably, the initially loaded routine, the first level user code, will load additional code and data. This first level user code may use the pre-initialized interface (UART or CAN) to receive data, a second level of code, and store it to arbitrary user-defined locations.

This second level of 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.

In all cases, the ST10F276 will still run in BSL mode, that is, with the watchdog timer disabled and limited access to the internal IFLASH area.

15.2.7 Exiting bootstrap loader mode

In order to execute a program in normal mode, the BSL mode must be terminated first. The ST10F276 exits BSL mode upon a software reset (level on P0L.4 is ignored) or a hardware reset (P0L.4 must be high in this case). After the reset, the ST10F276 will start executing

from location $00'0000_H$ of the internal Flash (User Flash) or the external memory, as programmed via pin $\overline{EA}$.

Note: If a bidirectional Software Reset is executed, and external memory boot is selected ($\overline{EA}$ = 0), a degeneration of the Software Reset event into a Hardware Reset can occur (Refer to section for details). This would imply that P0L.4 becomes transparent, so to exit from Bootstrap mode it would be necessary to release pin P0L.4 (it is no longer ignored).

15.2.8 Hardware requirements

Although the new bootstrap loader has been designed to be compatible with the old bootstrap loader, there are few hardware requirements related with the new bootstrap loader:

- External Bus configuration: need to have four segment address lines or less (keep CAN I/Os available);
- Usage of CAN pins (P4.5 and P4.6): even in bootstrap via UART, P4.5 (CAN1_RxD) cannot be used as Port output but only as input. The pin P4.6 (CAN1_TxD) can be used as input or output.
- Level on UART RxD and CAN1_RxD during the bootstrap phase (see *Figure 125* - Step 2): must be 1 (external pull-up's recommended).

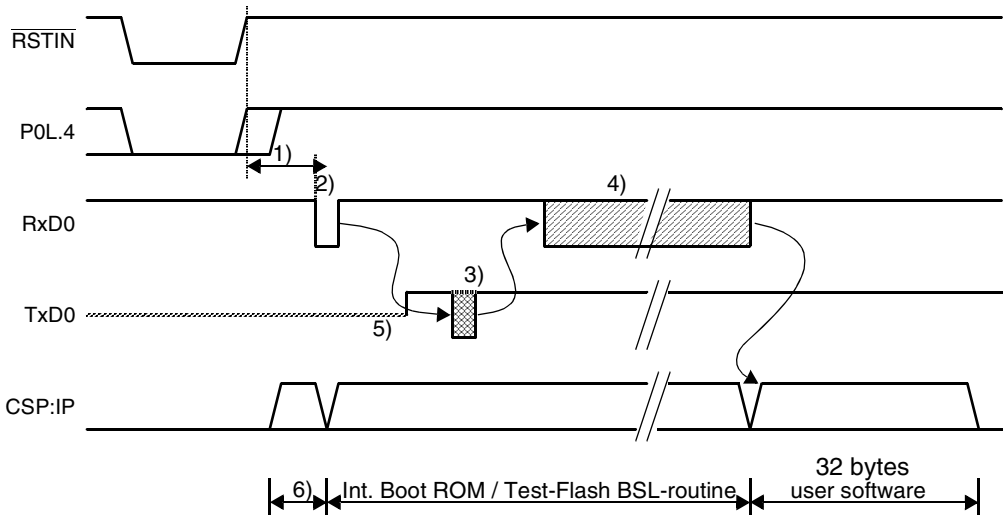
15.3 Standard bootstrap with UART (RS232 or K-Line)

15.3.1 Features

ST10F276 bootstrap via UART has the same overall behavior as the old ST10 bootstrap via UART:

- Same bootstrapping steps;
- Same bootstrap method: analyze the timing of a predefined byte, send back an acknowledge byte, load a fixed number of bytes and run then;
- Same functionalities: boot with different crystals and PLL ratios.

Figure 128. UART bootstrap loader sequence



1) BSL initialization time, > 1ms @ f_{CPU} = 40 MHz.

2) Zero byte (1 start bit, eight '0' data bits, 1 stop bit), sent by host.

3) Acknowledge byte, sent by ST10F276.

4) 32 bytes of code / data, sent by host.

5) Caution: TxD0 is only driven a certain time after reception of the zero byte (1.3ms @ f_{CPU} = 40 MHz).

6) Internal Boot ROM / Test-Flash.

15.3.2 Entering bootstrap via UART

The ST10F276 enters BSL mode, if pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Test-Flash, no part of the standard mask ROM or Flash memory area is required for this.

After entering BSL mode and the respective initialization the ST10F276 scans the RxD0 line to receive a zero byte, that is, one start bit, eight '0' data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface ASC0 accordingly and switches pin TxD0 to output. Using this baudrate, an acknowledge byte is returned to the host that provides the loaded data.

The acknowledge byte is **D5h** for the ST10F276.

15.3.3 ST10 configuration in UART BSL (RS232 or K-line)

When the ST10F276 has entered BSL mode on UART, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked in bold***):

Watchdog Timer:	Disabled
Register SYSCON:	***0400$_H$*** (1)
Context Pointer CP:	FA00$_H$
Register STKUN:	FA00$_H$
Stack Pointer SP:	***FA40$_H$***
Register STKOV:	FC00$_H$
Register BUSCON0:	according to startup configuration (1)
Register S0CON:	***8011$_H$***
Register BUSCON0:	according to startup configuration (2)
Register S0BG:	according to '00' byte
P3.10 / TXD0:	note (1)
DP3.10:	note (1)

1. In Bootstrap modes (standard or alternate) ROMEN, bit 10 of SYSCON, is always set regardless of EA pin level. BYTDIS, bit 9 of SYSCON, is set according to data bus width selection via Port0 configuration.
2. BUSCON0 is initialized with 0000h, external bus disabled, if pin EA is high during reset. If pin EA is low during reset, BUSACT0, bit 10, and ALECTL0, bit 9, are set enabling the external bus with lengthened ALE signal. BTYP field, bit 7 and 6, is set according to Port0 configuration.

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin TxD0 is configured as output, so the ST10F276 can return the acknowledge byte. Even if the internal IFLASH is enabled, no code can be executed out of it.

15.3.4 Loading the start-up code

After sending the acknowledge byte the BSL enters a loop to receive 32 bytes via ASC0. These bytes are stored sequentially into locations 00'FA40$_H$ through 00'FA5F$_H$ of the IRAM. So up to 16 instructions may be placed into the RAM area. To execute the loaded code the BSL then jumps to location 00'FA40$_H$, that is, the first loaded instruction. The bootstrap loading sequence is now terminated, the ST10F276 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 ST10F276 will still run in BSL mode, that is, with the watchdog timer disabled and limited access to the internal Flash area. All code fetches from the internal IFLASH area (01'0000$_H$...08'FFFF$_H$) are redirected to the special Test-Flash. Data read operations will access the internal Flash of the ST10F276, if any is available, but will return undefined data on ROM-less devices.

15.3.5 Choosing the baud rate for the BSL via UART

The calculation of the serial baudrate for ASC0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the ST10F276 with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to insure proper data transfer.

$$B_{ST10F276} = \frac{f_{CPU}}{32 \cdot (S0BRL + 1)}$$

The ST10F276 uses timer T6 to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the first deviation from the real baudrate, the next deviation is implied by the computation of the S0BRL reload value from the timer contents. The formula below shows the association:

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

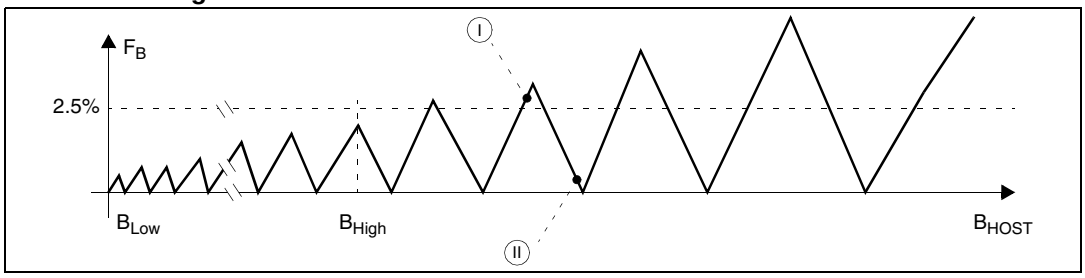
For a correct data transfer from the host to the ST10F276 the maximum deviation between the internal initialized baudrate for ASC0 and the real baudrate of the host should be below 2.5%. The deviation (F_B, in percent) between host baudrate and ST10F276 baudrate can be calculated via the formula below:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \cdot 100 \ \% \ , \quad F_B \leq 2.5 \ \%$$

Note: Function (F_B) does not consider the tolerances of oscillators and other devices supporting the serial communication.

This baudrate deviation is a nonlinear function depending on the CPU clock and the baudrate of the host. The maxima of the function (F_B) increases with the host baudrate due to the smaller baudrate pre-scaler factors and the implied higher quantization error (see *Figure 129*).

Figure 129. Baudrate deviation between host and ST10F276



The minimum baudrate (B_{Low} in *Figure 129*) is determined by the maximum count capacity of timer T6, when measuring the zero byte, that is, it depends on the CPU clock. Using the maximum T6 count 2^{16} in the formula the minimum baudrate can be calculated. The lowest standard baudrate in this case would be 1200 Baud. Baudrates below B_{Low} would cause T6 to overflow. In this case ASC0 cannot be initialized properly.

The maximum baudrate (B_{High} in *Figure 129*) is the highest baudrate where the deviation still does not exceed the limit, that is, all baudrates between B_{Low} and B_{High} are below the deviation limit. The maximum standard baudrate that fulfills this requirement is 19200 Baud.

Higher baudrates, however, may be used as long as the actual deviation does not exceed the limit. A certain baudrate (marked I) in *Figure 129* may for example, violate the deviation limit, while an even higher baudrate (marked II) in *Figure 129* stays very well below it. This depends on the host interface.

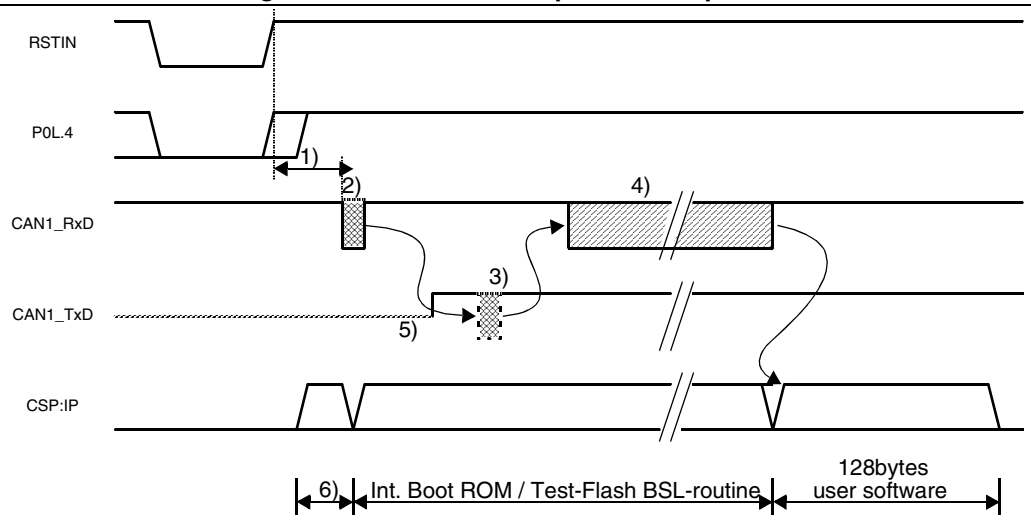
15.4 Standard bootstrap with CAN

15.4.1 Features

The bootstrap via CAN has the same overall behavior as the bootstrap via UART:

- Same bootstrapping steps;
- Same bootstrap method: analyze the timing of a predefined frame, send back an acknowledge frame **BUT only on request**, load a fixed number of bytes and run then;
- Same functionalities: boot with different crystals and PLL ratios.

Figure 130. CAN bootstrap loader sequence



1) BSL initialization time, > 1ms @ fCPU = 40 MHz.

2) Zero frame (CAN message: standard ID = 0, DLC = 0), sent by host.

3) CAN message (standard ID = E6h, DLC = 3, Data0 = D5h, Data1-Data2 = IDCHIP_low-high), sent by ST10F276 on request

4) 128bytes of code / data, sent by host.

5) Caution: CAN1_TxD is only driven a certain time after reception of the zero byte (1.3ms @ f_{CPU} = 40 MHz).

6) Internal Boot ROM / Test-Flash.

The Bootstrap Loader may be used to load the complete application software into ROM-less systems, it may load temporary software into complete systems for testing or calibration, it may also be used to load a programming routine for Flash devices.

The BSL mechanism may be used for standard system start-up as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

15.4.2 Entering the CAN bootstrap loader

The ST10F276 enters BSL mode, if pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Test-Flash, no part of the standard mask ROM or Flash memory area is required for this.

After entering BSL mode and the respective initialization the ST10F276 scans the CAN1_TxD line to receive the following initialization frame:

- standard identifier = 0h
- DLC = 0h

As all the bits to be transmitted are dominant bits, a succession of 5 dominant bits and 1 stuff bit on the CAN network is used. From the duration of this frame it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the CAN1 interface accordingly, switches pin CAN1_TxD to output and enables the CAN1 interface to take part in the network communication. Using this baudrate, an Message Object is configured in order to send an acknowledge frame. The ST10F276 will not send this Message Object but the host can request it by sending remote frame.

The acknowledge frame is the following for the ST10F276:

- standard identifier = E6h
- DLC = 3h
- Data0 = D5h, that is, generic acknowledge of the ST10 devices
- Data1 = IDCHIP least significant byte
- Data2 = IDCHIP most significant byte

For the ST10F276, IDCHIP = 114Xh.

Note: Two behaviors can be distinguished in the acknowledging of the ST10 to the host. If the host is behaving according to the CAN protocol, as at the beginning the ST10 CAN is not configured, the host will be alone on the CAN network and will not get any acknowledge. It will automatically resent the zero frame. As soon as the ST10 CAN will be configured, it will acknowledge the zero frame. The "acknowledge frame" with identifier 0xE6, will be configured, but the Transmit Request will not be set. The host can request this frame to be sent, and therefore get the IDCHIP, by sending a remote frame.

Flash programming software now knows immediately the exact type of device to be programmed. as the IDCHIP is sent in the acknowledge frame.

15.4.3 ST10 configuration in CAN BSL

When the ST10F276 has entered BSL mode via CAN, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked in bold***):

Watchdog Timer:	**Disabled**	
Register SYSCON:	***0404_H*** (1)	; XPEN bit set
Context Pointer CP:	$FA00_H$	
Register STKUN:	$FA00_H$	
Stack Pointer SP:	***$FA40_H$***	
Register STKOV:	$FC00_H$	
Register BUSCON0:	acc. to start-up config.(2)	
CAN1 Status/Control Register:	0000_H	
CAN1 Bit Timing Register:	according to '0' frame	
Register XPERCON:	$042D_H$	; XRAM1-2, CAN1, XFlash and XMISC enabled
P4.6 / CAN1_TxD:	note (1)	
DP4.6:	note (1)	

1. In Bootstrap modes (standard or alternate) ROMEN, bit 10 of SYSCON, is always set regardless of EA pin level. BYTDIS, bit 9 of SYSCON, is set according to data bus width selection via Port0 configuration.
2. BUSCON0 is initialized with 0000h, external bus disabled, if pin EA is high during reset. If pin EA is low during reset, BUSACT0, bit 10, and ALECTL0, bit 9, are set enabling the external bus with lengthened ALE signal. BTYP field, bit 7 and 6, is set according to Port0 configuration.

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin CAN1_TxD1 is configured as output, so the ST10F276 can return the identification frame. Even if the internal IFlash is enabled, no code can be executed out of it.

15.4.4 Loading the start-up code via CAN

After sending the acknowledge byte the BSL enters a loop to receive 128 bytes via CAN1.

Hint: the number of bytes loaded when booting via the CAN interface has been extended to 128 bytes in order to allow the re-configuration of the CAN Bit Timing Register with the best timings (synchronization window, ...). This can be achieved by the following sequence of instructions:

```
ReconfigureBaudRate:
   MOV   R1,#041h
   MOV   DPP3:0EF00h,R1    ; Put CAN in Init, enable Configuration Change
   MOV   R1,#01600h
   MOV   DPP3:0EF06h,R1    ; 1MBaud at Fcpu = 20 MHz
```

These 128 bytes are stored sequentially into locations 00'$FA40_H$ through 00'$FABF_H$ of the IRAM. So up to 64 instructions may be placed into the RAM area. To execute the loaded code the BSL then jumps to location 00'$FA40_H$, that is, the first loaded instruction. The bootstrap loading sequence is now terminated, the ST10F276 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 64 instructions. This second receive loop may directly use the pre-initialized CAN interface 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.

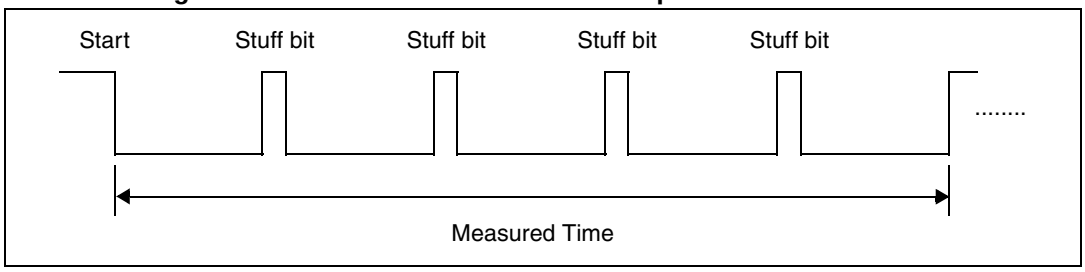
This process may go through several iterations or may directly execute the final application. In all cases the ST10F276 will still run in BSL mode, that is, with the watchdog timer disabled and limited access to the internal Flash area. All code fetches from the internal Flash area ($01'0000_H$...$08'FFFF_H$) are redirected to the special Test-Flash. Data read operations will access the internal Flash of the ST10F276, if any is available, but will return undefined data on ROM-less devices.

15.4.5 Choosing the baudrate for the BSL via CAN

The Bootstrap via CAN acts in the same way than the UART bootstrap mode. When the ST10F276 is started in BSL mode, it polls the RxD0 and CAN1_RxD lines. On polling a low level on one of these lines, a timer is launched that will be stopped when the line gets back to high level.

For CAN communication, the algorithm is made to receive a zero frame, i-e standard identifier is 0x0, DLC is 0. This frame will produce on the network the following levels: 5D, 1R, 5D, 1R, 5D, 1R, 5D, 1R, 5D, 1R, 4D, 1R, 1D, 11R. The algorithm will let run the timer until the detection of the 5th recessive bit. This way the bit timing is calculated over the duration of 29 bit time: this is done in order to minimize the error introduced by the polling.

Figure 131. Bitrate measurement over a predefined zero-frame



Error induced by the polling

The code used for the polling is the following:

```
WaitCom:
   JNB    P4.5,CAN_Boot           ; if SOF detected on CAN, then go to CAN
                                  ; loader
   JB     P3.11,WaitCom           ; Wait for start bit at RxD0
   BSET   T6R                     ; Start Timer T6
....
CAN_Boot:
   BSET   PWMCON0.0               ; Start PWM Timer0
                                   ; (resolution is 1 CPU clk cycle)
   JMPR   cc_UC,WaitRecessiveBit
WaitDominantBit:
   JB     P4.5,WaitDominantBit    ; wait for end of stuff bit
WaitRecessiveBit:
   JNB    P4.5,WaitRecessiveBit; wait for 1st dominant bit = Stuff bit
   CMPI1 R1,#5                    ; Test if 5th stuff bit detected
   JMPR   cc_NE,WaitDominantBit   ; No, go back to count more
   BCLR   PWMCON.0                ; Stop timer
                                  ; here the 5th stuff bit is detected:
                                  ; PT0 = 29 Bit_Time (25D and 4R)
```

Therefore the maximum error at the detection of the communication on CAN pin is:

(1 not taken + 1 taken jumps) + 1 taken jump + 1 bit set: (6) + 6 CPU clock cycles

The error at the detection for the 5th recessive bit is:

(1 taken jump) + 1 not taken jump + 1 compare + 1 bit clear: (4) + 6 CPU cycles

In the worst case the induced error is of six CPU clock cycles. So the polling could induce an error of six timer ticks.

Error induced by the Baud rate calculation

The content of the timer PT0 counter corresponds to 29 bit times. This gives the following equation:

PT0 = 58 x (BRP + 1) X (1 + Tseg1 + Tseg2)

where BRP, Tseg1 and Tseg2 are the field of the CAN Bit Timing register.

The CAN protocol specification recommends to implement a bit time composed by at least 8 time quantum (tq). This recommendation have been applied here. Moreover, the maximum bit time length is 25 tq. In order to have a good precision, the target is to have the smallest Bitrate Prescaler (BRP) and the maximum number of tq in a bit time.

This gave the following ranges for PT0 according to BRP:

$8 \leq 1 + Tseg1 + Tseg2 \leq 25$

$464 \times (1 + BRP) \leq PT0 \leq 1450 \times (1 + BRP)$

Table 46. Ranges of timer contents versus BRP value

BRP	PT0_min	PT0_max	Comments
0	464	1450	
1	1451	2900	
2	2901	4350	
3	4351	5800	
4	5801	7250	
5	7251	8700	
..	..	..	
43	20416	63800	
44	20880	65250	
45	21344	66700	Possible timer overflow
..	..	..	
63	X	X	

The error coming from the measurement of the 29 Bit is:

$e_1 = 6 / [PT0]$

It is maximal for the smallest BRP value and the smallest number of ticks in PT0. Therefore:

$e_{1\ Max} = 1.29\%$

To have a better precision, the target is to have the smallest BRP so that the time quantum is the smallest possible. Thus an error on the calculation of time quanta in a bit time is minored.

In order to do so, the value of PT0 is divided in ranges of 1450 ticks. In the algorithm, PT0 is divided by 1451 and the result is BRP.

The calculated BRP value is then used to divide PT0 in order to have the value of *(1 + Tseg1 + Tseg2)*. A table is made to set the values for *Tseg1* and *Tseg2* according to the value of *(1 + Tseg1 + Tseg2)*. These values of *Tseg1* and *Tseg2* are chosen in order to reach a sample point between 70% and 80% of the bit time.

During the calculation of *(1 + Tseg1 + Tseg2)*, an error e_2 can be introduced by the division. This error is of 1 time quantum maximum.

To compensate any possible error on bitrate, the (Re)Synchronization Jump Width is fixed to 2 time quanta.

15.4.6 How to compute the baud rate error

Considering the following conditions, a computation of the error is reported as example.

- CPU frequency: 20 MHz
- Target bitrate: 1Mbit/s

In these conditions, the content of PT0 timer for 29 bits should be:

$$[PT0] = \frac{29 \times Fcpu}{Bitrate} = \frac{29 \times 20 \times 6}{1 \times 10^6} = 580$$

Therefore:

574 < [PT0] < 586

This gives:

- BRP = 0
- tq = 100 ns

Computation of *1 + Tseg1 + Tseg2*: considering the equation:

[PT0] = 58 x (1 + BRP) x (1 + Tseg1 + Tseg2)

Thus:

$$9 = \frac{574}{58} \leq Tseg1 + Tseg2 \leq \frac{586}{58} = 10$$

In the algorithm, a rounding to the superior value is made if the remainder of the division is greater than half of the divisor. Here it would have been the case if the PT0 content was 574. Thus in this example it results *1+Tseg1+Tseg2 = 10*, giving a bit time of exactly 1µs => no error in bitrate.

Note: In most cases (24 MHz, 32 MHz, 40 MHz of CPU frequency and 125, 250, 500 or 1 Mbit/s of bitrate) there is no error. Anyway, it is better to check the error with the real application parameters.

The content of the Bit Timing register will be: *0x1640*. This gives a sample point at 80%.

Note: The (Re)Synchronization Jump Width is fixed to 2 time quanta.

15.4.7 Bootstrap via CAN

After the bootstrap phase, ST10F276x CAN module is configured as follow:

- The pin P4.6 is configured as output (the latch value is '1' = recessive) to assume CAN1_TxD function.
- The MO2 is configured to output the acknowledge of the bootstrap with the standard identifier E6h, a DLC of 3 and Data0 = D5h, Data1&2 = IDCHIP.
- The MO1 is configured to receive messages with the standard identifier 5h. Its acceptance mask is set in order that all bits must match. The DLC received is not checked: the ST10 expects only 1 byte of data at a time.

No other message is sent by the ST10F276 after the acknowledge.

Note: The CAN boot waits for 128 byte of data instead of 32 (see UART boot). This is done in order to allow the User to reconfigure the CAN Bitrate as soon as possible.

15.5 Comparing the old and the new bootstrap loader

The following tables summarize the differences between the old ST10 (boot via UART only) bootstrap and the new one (boot via UART or CAN).

15.5.1 Software aspects

Table 47. Software topics summary

Previous bootstrap loader	ST10F27x bootstrap loader	Comments
Uses only 32 bytes in Dual-Port RAM from 00'FA40h	Uses up to 128 bytes in Dual-Port RAM from 00'FA40h	For compatibility between boot via UART and boot via CAN1, avoid loading the application software in the 00'FA60h/00'FABFh range.
Load 32 bytes from UART	Load 32 bytes from UART (boot via UART mode)	Same files can be used for boot via UART.
User selected X-Peripherals can be enabled during boot (step 3 or step 4)	X-Peripherals selection is fixed.	You can change the X-Peripherals selections through a specific code.

As the CAN1 is needed, XPERCON register is configured by the bootstrap loader code and bit XPEN of SYSCON is set. Anyway, as long as the EINIT instruction is not executed (and it

is not in the bootstrap loader code), the settings can be modified. The following steps must be performed in order to do this:

- disable the XPeripherals by clearing XPEN in SYSCON register. Attention: this part of code must not be located in XRAM as it will be disabled.
- enabled the needed XPeripherals by writing the correct value in XPERCON register.
- set XPEN bit in SYSCON.

15.5.2 Hardware aspects

Although the new bootstrap loader has been designed to be compatible with the old bootstrap loader, there are few hardware requirements with the new bootstrap loader hereafter summarized.

Table 48. Hardware topics summary

Actual bootstrap loader	New bootstrap loader	Comments
P4.5 can be used as output in BSL mode.	P4.5 cannot be used as user output in BSL mode, but only as CAN1_RxD or input or address-segments.	
Level on CAN1_RxD can change during boot step2.	Level on CAN1_RxD must be stable at '1' during boot step2.	External pull-up on P4.5 needed.

15.6 Alternate boot mode (ABM)

15.6.1 Activation

Alternate boot is activated with the combination '01' on Port0L[5..4] at the rising edge of RSTIN.

15.6.2 Memory mapping

ST10F276 has the same memory mapping for standard boot mode and for alternate boot mode:

- Test-Flash: mapped from 00'0000h. The Standard Bootstrap Loader can be started by executing a jump to the address of this routine (JMPS 00'xxxx; *address to be defined*).
- User Flash: the User Flash is divided in two parts: the IFLASH, visible only for memory reads and memory writes (no code fetch) and the XFLASH, visible for any ST10 access (memory read, memory write, code fetch).
- All ST10F276 XRAM and Xperipherals modules can be accessed if enabled in XPERCON register.

Note: The alternate boot mode can be used to reprogram the whole content of ST10F276 User Flash (except Block 0 in Bank 2, where the alternate boot is mapped into).

15.6.3 Interrupts

ST10 interrupt vector table is always mapped from address 00'0000h.

As a consequence, interrupts are not allowed in Alternate Boot Mode; all maskable and non maskable interrupts should be disabled.

15.6.4 ST10 configuration in alternate boot mode

When the ST10F276 has entered BSL mode via CAN, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked in bold***):

Watchdog Timer:	Disabled	
Register SYSCON:	***0404_H*** (1)	; XPEN bit set
Context Pointer CP:	$FA00_H$	
Register STKUN:	$FA00_H$	
Stack Pointer SP:	***$FA40_H$***	
Register STKOV:	$FC00_H$	
Register BUSCON0:	acc. to start-up config. (2)	
Register XPERCON:	$002D_H$	*; XRAM1, XRAM2, CAN1 and XFlash* enabled

1. In Bootstrap modes (standard or alternate) ROMEN, bit 10 of SYSCON, is always set regardless of EA pin level. BYTDIS, bit 9 of SYSCON, is set according to data bus width selection via Port0 configuration.
2. BUSCON0 is initialized with 0000h, external bus disabled, if pin EA is high during reset. If pin EA is low during reset, BUSACT0, bit 10, and ALECTL0, bit 9, are set enabling the external bus with lengthened ALE signal. BTYP field, bit 7 and 6, is set according to Port0 configuration.

Even if the internal IFlash is enabled, no code can be executed out of it.

Note: As the XFlash is needed, XPERCON register is configured by the ABM loader code and bit XPEN of SYSCON is set. Anyway, as long as the EINIT instruction is not executed (and it is not in the bootstrap loader code), the settings can be modified. The following steps must be performed in order to do this:

1. Copy in DPRAM a function that will do the following steps:
   a) Disable the XPeripherals by clearing XPEN in SYSCON register
   b) enabled the needed XPeripherals by writing the correct value in XPERCON register
   c) set XPEN bit in SYSCON
   d) return to calling address
2. Call the function from XFlash

The changing of XPERCON value cannot be executed from the XFlash because the XFlash will be disabled by the clearing of XPEN bit in SYSCON.

15.6.5 Watchdog

As for standard boot, the Watchdog timer should remain disabled during Alternate Boot Mode. In case of a Watchdog reset occurs, a software reset will be generated.

Note: See note in Section 15.2.7: Exiting bootstrap loader mode concerning software reset.

15.6.6 Exiting alternate boot mode

Once the ABM mode is entered, it can be exited only with a software or hardware reset.

Note: See note in Section 15.2.7: Exiting bootstrap loader mode concerning software reset.

15.6.7 Alternate boot user software

If the rules described here before are met (that is,: mapping of variables, disabling of interrupts, exit conditions, predefined vectors in Block 0 of Bank 2, Watchdog usage) then users can write the software they want to execute in this mode starting from 09’0000h.

15.6.8 User/alternate mode signature integrity check

The behavior of the Alternate Boot Mode is based on the computing of a signature between the content of two memory locations and a comparison with a reference signature. This requires that users who use Alternate Boot have reserved and programmed the Flash memory locations according to:

User mode signature

- 00'0000h: memory address of *operand0* for the signature computing
- 00’1FFCh: memory address of *operand1* for the signature computing
- 00’1FFEh: memory address for the signature reference

Alternate mode signature

- 09'0000h: memory address of *operand0* for the signature computing
- 09’1FFCh: memory address of *operand1* for the signature computing
- 09’1FFEh: memory address for the signature reference

The value for *operand0, operand1* and the signature should be such that the following sequence should be successfully executed:

```
MOV    Rx, CheckBlock1Addr; 00`0000h for standard reset
ADD    Rx, CheckBlock2Addr; 00`1FFCh for standard reset
CPLB   RLx                ; 1s complement of the lower
                          ; byte of the sum
CMP    Rx, CheckBlock3Addr; 00`1FFEh for standard reset
```

15.6.9 Alternate boot user software aspects

User defined alternate boot code must start at 09’0000h. A new SFR has been created on ST10F276 to indicate that the device is running in Alternate Boot Mode: bit 5 of EMUCON (mapped at 0xFE0Ah) is set when the alternate boot is selected by the reset configuration. All the other bits should be ignored when checking the content of this register to read the value of bit5.

This bit is a read only bit. It remains set until next software or hardware reset.

15.6.10 EMUCON register

EMUCON (FE0Ah / 05h) SFR Reset Value: - - xxh

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

Bit	Function
ABM	**ABM Flag (or TMOD3)** '0': Alternate Boot Mode is not selected by reset configuration on P0L[5..4] '1': Alternate Boot Mode is selected by reset configuration on P0L[5..4]: this bit is set if P0L[5..4] = '01' during hardware reset.

15.6.11 Internal decoding of test modes

The test mode decoding logic is located inside ST10F276 Bus Controller.

The decoding is as follow:

- Alternate Boot Mode decoding: ($\overline{\text{P0L.5}}$ & P0L.4)
- Standard Bootstrap decoding: (P0L.5 & $\overline{\text{P0L.4}}$)
- Normal operation: (P0L.5 & P0L.4)

The other configurations select ST internal test modes.

15.6.12 Example

In the following example Alternate Boot Mode works as follow:

- On rising edge of $\overline{\text{RSTIN}}$ pin, the reset configuration is latched.
- if Bootstrap Loader mode is not enabled (P0L[5..4] = '11'), then ST10F276 hardware proceeds with a standard hardware reset procedure.
- If standard Bootstrap Loader is enabled (P0L[5..4] = '10'), then the standard ST10 Bootstrap Loader is enabled and a variable is cleared to indicate that ABM is not enabled.
- If Alternate Boot Mode is selected (P0L[5..4] = '01'), then depending on signatures integrity checks a predefined reset sequence is activated.

15.7 Selective boot mode

The Selective Boot is a sub-case of the Alternate Boot Mode. When none of the signatures is correct, instead of executing the Standard Bootstrap Loader (triggered by P0L.4 low at reset), additional check is made.

Address 00'1FFCh is read again with the following behavior:

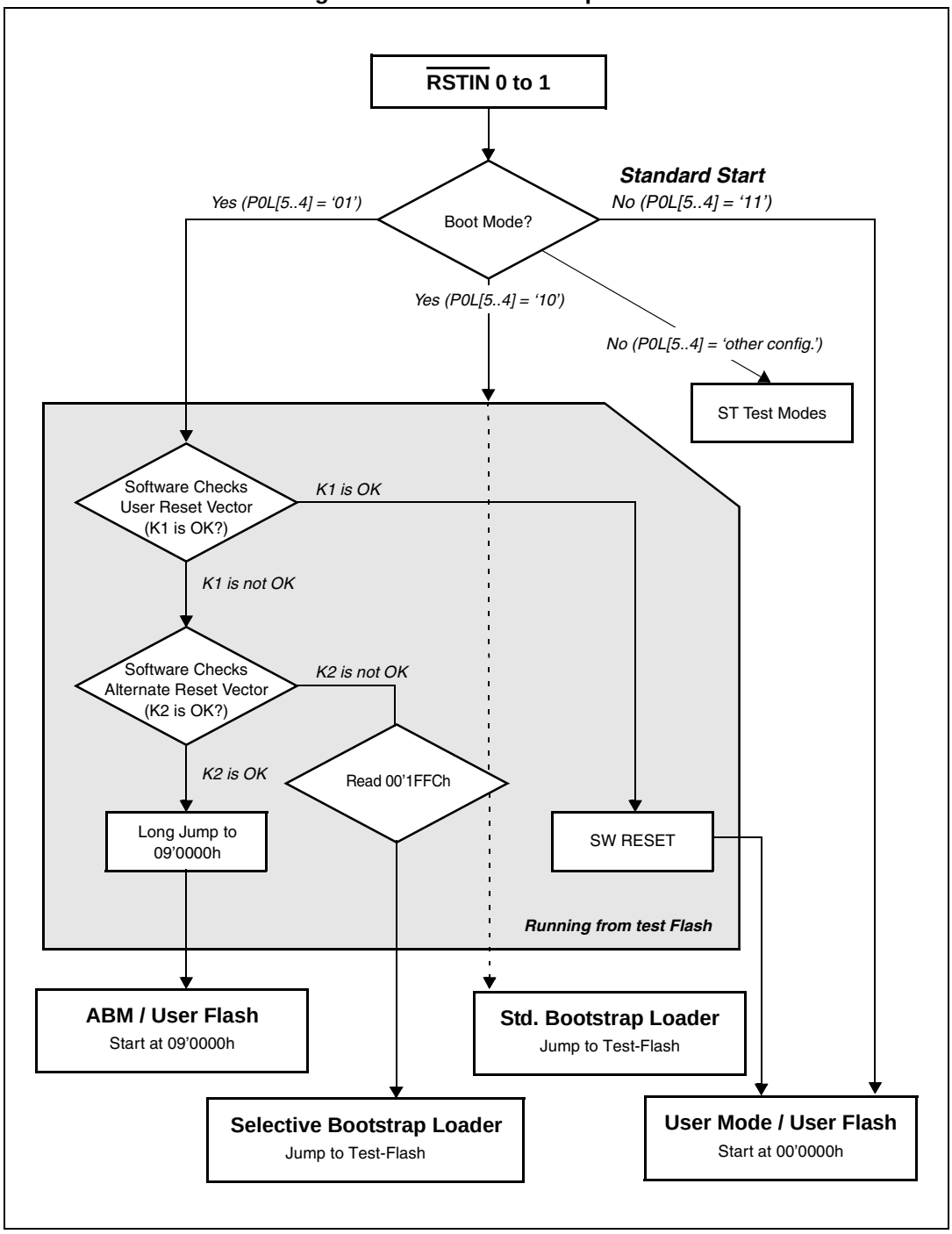
- If value is 0000h or FFFFh, then a jump is performed to the Standard Bootstrap Loader.
- Else:
  - High byte is disregarded.
  - Low byte bits selects which communication channel is enabled.

Bit	Function
0	**UART Selection** '0': UART will not be watched for a Start condition. '1': UART will be watched for a Start condition.
1	**CAN1 Selection** '0': CAN1 will not be watched for a Start condition. '1': CAN1 will be watched for a Start condition.
2..7	**Reserved** For upward compatibility, must be programmed to '0'

Therefore the value:

- 0xXX03 will configure the Selective Bootstrap Loader to poll for RxD0 and CAN1_RxD.
- 0xXX01 will configure the Selective Bootstrap loader to poll only RxD0 (no boot via CAN).
- 0xXX02 will configure the Selective Bootstrap Loader to poll only CAN1_RxD (no boot via UART).
- other values will let the ST10F276 executing an endless loop into the Test-Flash.

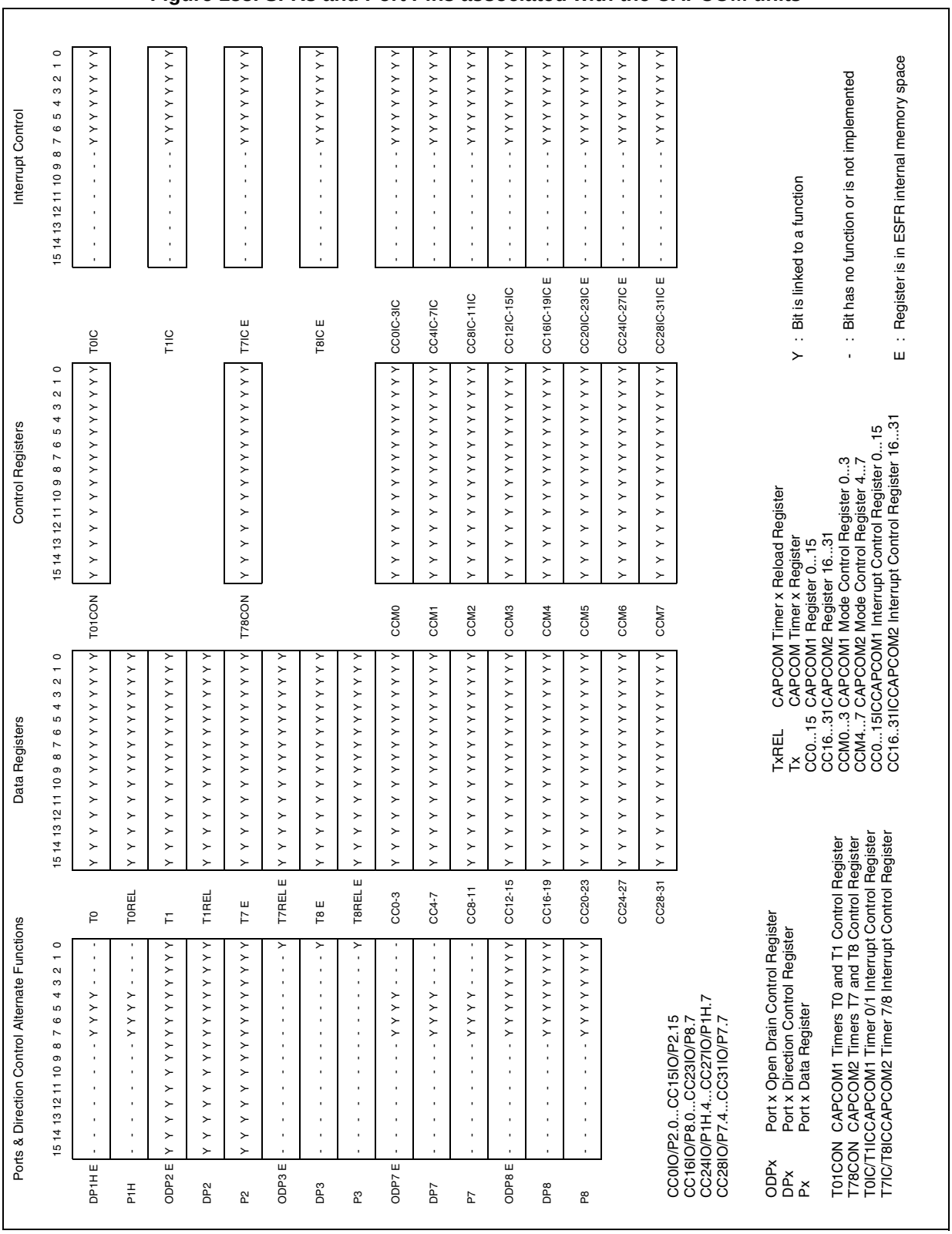
Figure 132. Reset boot sequence



16 The capture / compare units

The ST10F276 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 of up to 16 channels per unit with a minimum of software intervention. For programming, the term 'CAPCOM unit' refers to a set of SFRs associated to the peripheral, including the port pins which may be used for alternate input / output functions including their direction control bits.

Figure 133. SFRs and Port Pins associated with the CAPCOM units



A CAPCOM unit handles high speed I/O tasks such as pulse and waveform generation, pulse width modulation, or recording of the time at which specific events occur. It also allows the implementation of up to 16 software timers. The maximum resolution of the CAPCOM units is calculated with the formula in *Section 16.1: CAPCOM timers on page 329* and is specified in the device datasheet.

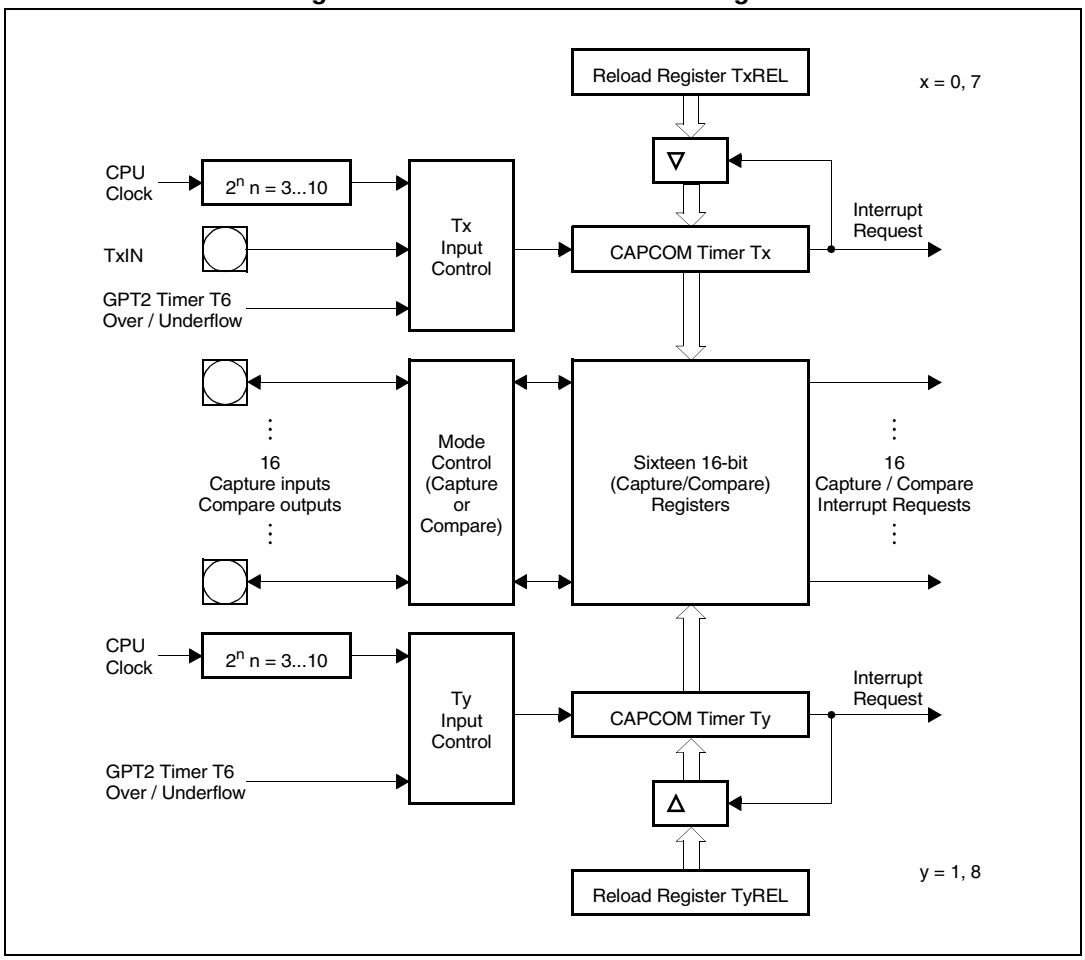
Each CAPCOM unit consists of two 16-bit timers (T0 / T1 in CAPCOM1, T7 / T8 in CAPCOM2), each with its own reload register (TxREL), and a bank of sixteen dual purpose 16-bit capture / compare registers (CC0 through CC15 in CAPCOM1, CC16 through CC31 in CAPCOM2).

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 134 shows the basic structure of the two CAPCOM units.

Figure 134. CAPCOM unit block diagram



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

16.1 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 135. Block diagram of CAPCOM timers T0 and T7

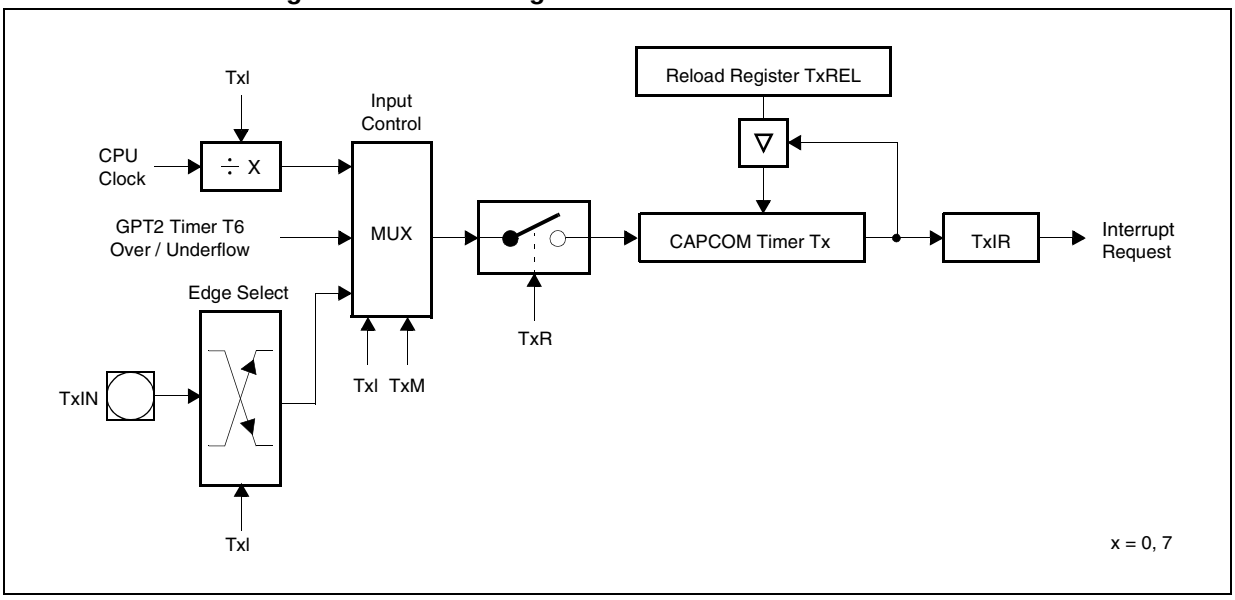
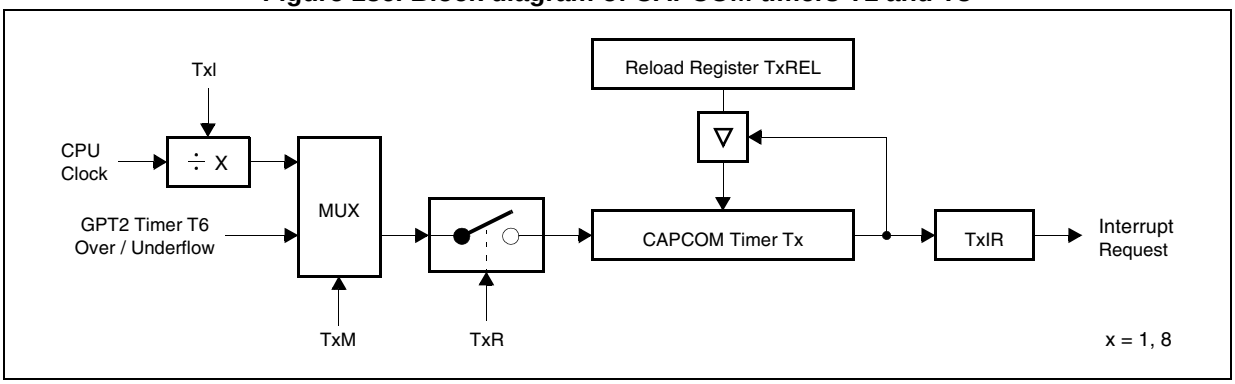


Figure 136. Block diagram of CAPCOM timers T1 and T8



Note: When an external input signal is connected to the input lines of both T0 and T7, these timers count the input signal synchronously. Thus the two timers can be regarded as one timer whose contents can be compared with 32 capture registers.

The functions of the CAPCOM timers are controlled via the bit-addressable 16-bit control registers T01CON and T78CON. The high-byte of T01CON controls T1, the low-byte of T01CON controls T0, the high-byte of T78CON controls T8, the low-byte of T78CON controls T7. The control options are identical for all four timers (except for external input).

T01CON (FF50h / A8h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	T1R	-	-	T1M	T1I			-	T0R	-	-	T0M	T0I		
	RW			RW	RW				RW			RW	RW		

T78CON (FF20h / 90h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	T8R	-	-	T8M	T8I			-	T7R	-	-	T7M	T7I		
	RW			RW	RW				RW			RW	RW		

Bit	Function
TxI	**Timer / Counter x Input Selection** Timer Mode (TxM = '0') Input Frequency = $f_{CPU} / 2^{[(TxI)+3]}$ See also table below for examples. Counter Mode (TxM = '1'): X00 Overflow / Underflow of GPT2 Timer 6 X01 Positive (rising) edge on pin TxIN 1) X10 Negative (falling) edge on pin TxIN 1) X11 Any edge (rising and falling) on pin TxIN 1)
TxM	**Timer / Counter x Mode Selection** '0': Timer Mode (Input derived from internal clock) '1': Counter Mode (Input from External Input or T6)
TxR	**Timer / Counter x Run Control** '0': Timer / Counter x is disabled '1': Timer / Counter x is enabled

Note: This selection is available for timers T0 and T7. Timers T1 and T8 will stop at this selection!

The run flags T0R, T1R, T7R and T8R enable or disable the timers. The following description of the timer modes and operation always applies to the enabled state of the timers, the respective run flag is assumed to be set to '1'.

In all modes, the timers are always counting upward. The current timer values are accessible for the CPU in the timer registers Tx, which are non bit-addressable SFRs. When the CPU writes to a register Tx in the state immediately before the respective timer increment, a reload is to be performed, the CPU write operation has priority and the increment or reload is disabled to 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 eight CPU clock cycles before it changes its level again. The incremented count value appears in SFR T0 / T7 within eight 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 overflows 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, from the reloaded value.

The reload registers TxREL are not bit-addressable.

16.2 CAPCOM unit timer interrupts

Upon a timer overflow the corresponding timer interrupt request flag TxIR for the respective timer will be set. This flag can be used to generate an interrupt or trigger a PEC service request, when enabled by the respective interrupt enable bit TxIE.

Each timer has its own bit-addressable interrupt control register (TxIC) and its own interrupt vector (TxINT). The organization of the interrupt control registers TxIC is identical with the other interrupt control registers.

T0IC (FF9Ch / CEh) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T0IR	T0IE	ILVL				GLVL	
								RW	RW	RW				RW	

T1IC (FF9Eh / CFh) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T1IR	T1IE	ILVL				GLVL	
								RW	RW	RW				RW	

T7IC (F17Ah / BEh) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T7IR	T7IE	ILVL				GLVL	
								RW	RW	RW				RW	

T8IC (F17Ch / BFh) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T8IR	T8IE	ILVL				GLVL	
								RW	RW	RW				RW	

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

16.3 Capture / compare registers

The 16-bit capture / compare registers CC0 through CC31 are used as data registers for capture or compare operations with respect to timers T0 / T1 and T7 / T8. The capture / compare registers are not bit-addressable.

Each of the registers CC0...CC31 may be individually programmed for capture mode or one of four different compare modes (except for CC24...CC27), and may be allocated individually to one of the two timers of the respective CAPCOM unit (T0 or T1, and T7 or T8, respectively). A special combination of compare modes additionally allows the implementation of a 'double-register' compare mode.

When capture or compare operation is disabled for one of the CCx registers, it may be used for general purpose variable storage.

The functions of the 32 capture / compare registers are controlled by the eight mode control registers named CCM0...CCM7 which are all organized identically (see description below). These 16-bit registers are bit-addressable.

Each register contains bit for mode selection and timer allocation of four capture / compare registers.

Capture / compare mode registers for the CAPCOM1 unit (CC0...CC15)

CCM0 (FF52h / A9h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC3		CCMOD3		ACC2		CCMOD2		ACC1		CCMOD1		ACC0		CCMOD0	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM1 (FF54h / AAh) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC7		CCMOD7		ACC6		CCMOD6		ACC5		CCMOD5		ACC4		CCMOD4	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM2 (FF56h / ABh) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC11		CCMOD11		ACC10		CCMOD10		ACC9		CCMOD9		ACC8		CCMOD8	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM3 (FF58h / ACh) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC15		CCMOD15		ACC14		CCMOD14		ACC13		CCMOD13		ACC12		CCMOD12	
RW		RW		RW		RW		RW		RW		RW		RW	

Capture / compare mode registers for the CAPCOM2 unit (CC16...CC31)

CCM4 (FF22h / 91h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC19		CCMOD19		ACC18		CCMOD18		ACC17		CCMOD17		ACC16		CCMOD16	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM5 (FF24h / 92h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC23		CCMOD23		ACC22		CCMOD22		ACC21		CCMOD21		ACC20		CCMOD20	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM6 (FF26h / 93h) SFR Reset Value: 0000h

15	14 13 12	11	10 9 8	7	6 5 4	3	2 1 0
ACC27	CCMOD27	ACC26	CCMOD26	ACC25	CCMOD25	ACC24	CCMOD24
RW	RW	RW	RW	RW	RW	RW	RW

CCM7 (FF28h / 94h) SFR Reset Value: 0000h

15	14 13 12	11	10 9 8	7	6 5 4	3	2 1 0
ACC31	CCMOD31	ACC30	CCMOD30	ACC29	CCMOD29	ACC28	CCMOD28
RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
CCMODx	**Mode Selection for Capture / Compare Register CCx** The available capture / compare modes are listed in the table below.
ACCx	**Allocation bit for Capture / Compare Register CCx** ‘0’: CCx allocated to Timer T0 (CAPCOM1) / Timer T7 (CAPCOM2) ‘1’: CCx allocated to Timer T1 (CAPCOM1) / Timer T8 (CAPCOM2)

16.3.1 Selection of capture modes and compare modes

CCMODx	Selected Operating Mode
0 0 0	**Disable Capture and Compare Modes** The respective CAPCOM register may be used for general variable storage.
0 0 1	Capture on Positive Transition (Rising Edge) at Pin CCxIO
0 1 0	Capture on Negative Transition (Falling Edge) at Pin CCxIO
0 1 1	Capture on Positive and Negative Transition (Both Edges) at Pin CCxIO
1 0 0	**Compare Mode 0: Interrupt Only** Several interrupts per timer period. Enables double-register compare mode for registers CC8...CC15 and CC24...CC31.
1 0 1	**Compare Mode 1: Toggle Output Pin on each Match** Several compare events per timer period. This mode is required for double-register compare mode for registers CC0...CC7 and CC16...CC23.
1 1 0	**Compare Mode 2: Interrupt Only** Only one interrupt per timer period.
1 1 1	**Compare Mode 3: Set Output Pin on each Match** Reset output pin on each timer overflow. Only one interrupt per timer period.

The detailed discussion of the capture and compare modes is valid for all the capture / compare channels, so registers, bits and pins are only referenced by the place holder ‘x’.

Note: Capture / compare channels 24...27 generate an interrupt request but do not provide an output signal. The resulting exceptions are indicated in the following subsections.
A capture or compare event on channel 31 may be used to trigger a channel injection on the ST10F276’s A / D converter if enabled.

16.4 Capture mode

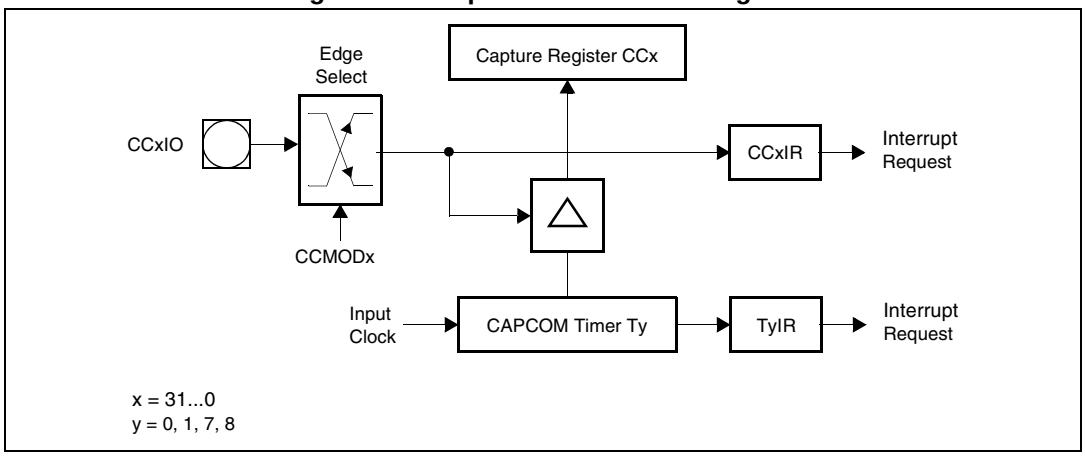
In response to an external event the content of the associated timer (T0 / T1 or T7 / T8, depending on the used CAPCOM unit and the state of the allocation control bit ACCx) is latched into the respective capture register CCx. The external event causing a capture can be programmed to be either a positive, 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 137*).

In order to use the respective port pin as an 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 eight CPU clock cycles before it changes its level.

During these eight 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 137. Capture mode block diagram



16.5 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 eight 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 49. 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.

16.5.1 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’.

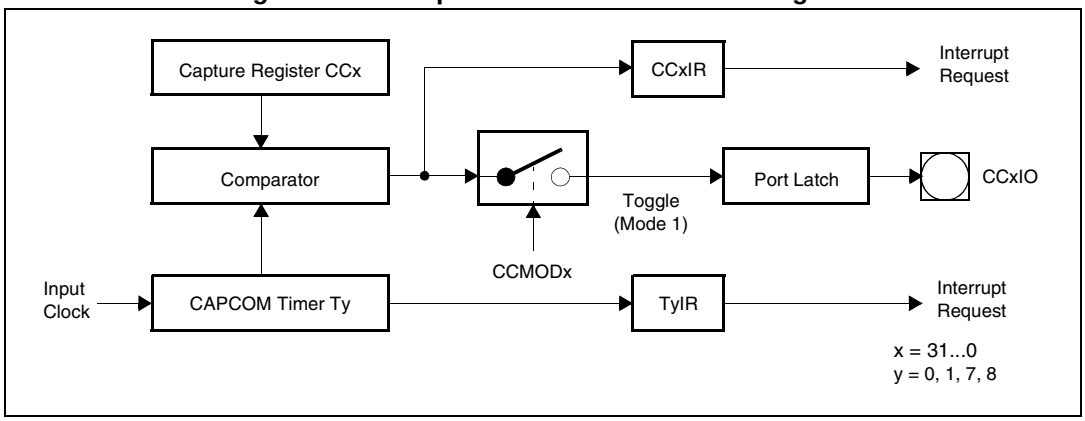
In this mode, the interrupt request flag CCxIR is set each time a match is detected between the content of compare register CCx and the allocated timer.

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 *Chapter 16.5.5*).

Figure 138. 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 139*).

16.5.2 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 139*).

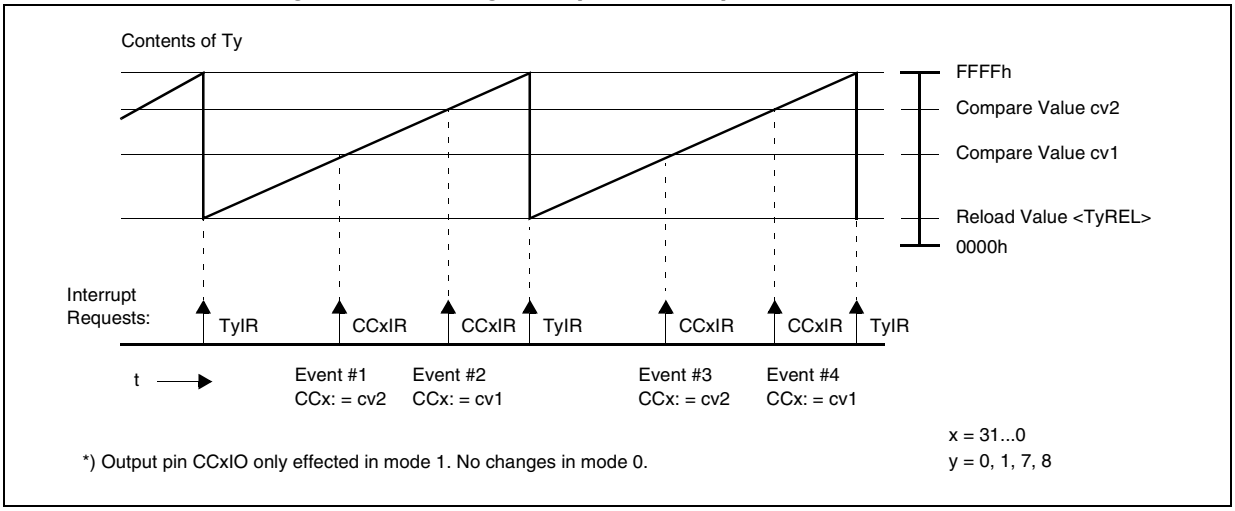
Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.

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 *Section 16.5.5: 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 139. Timing example for compare modes 0 and 1



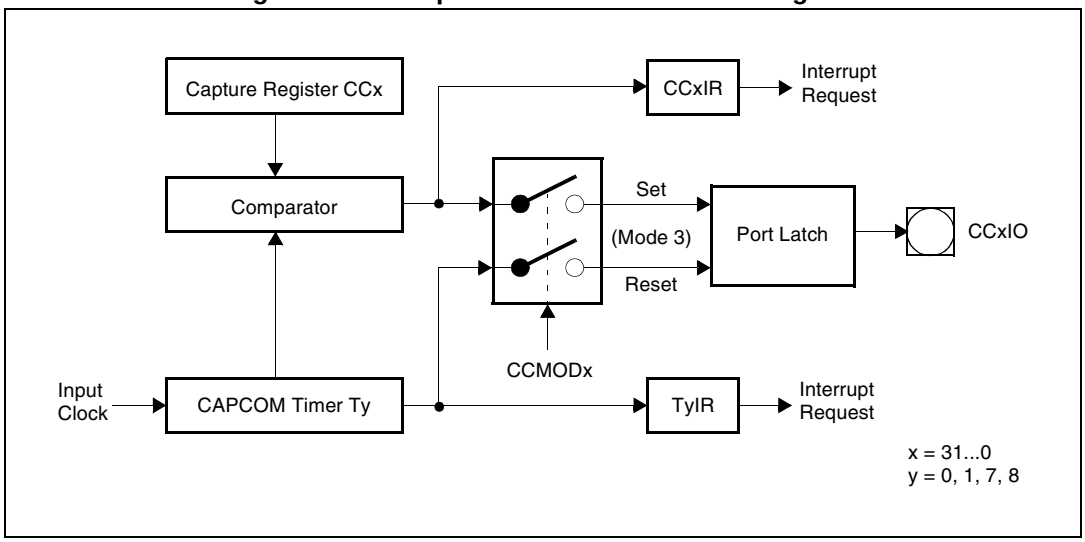
16.5.3 Compare mode 2

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.

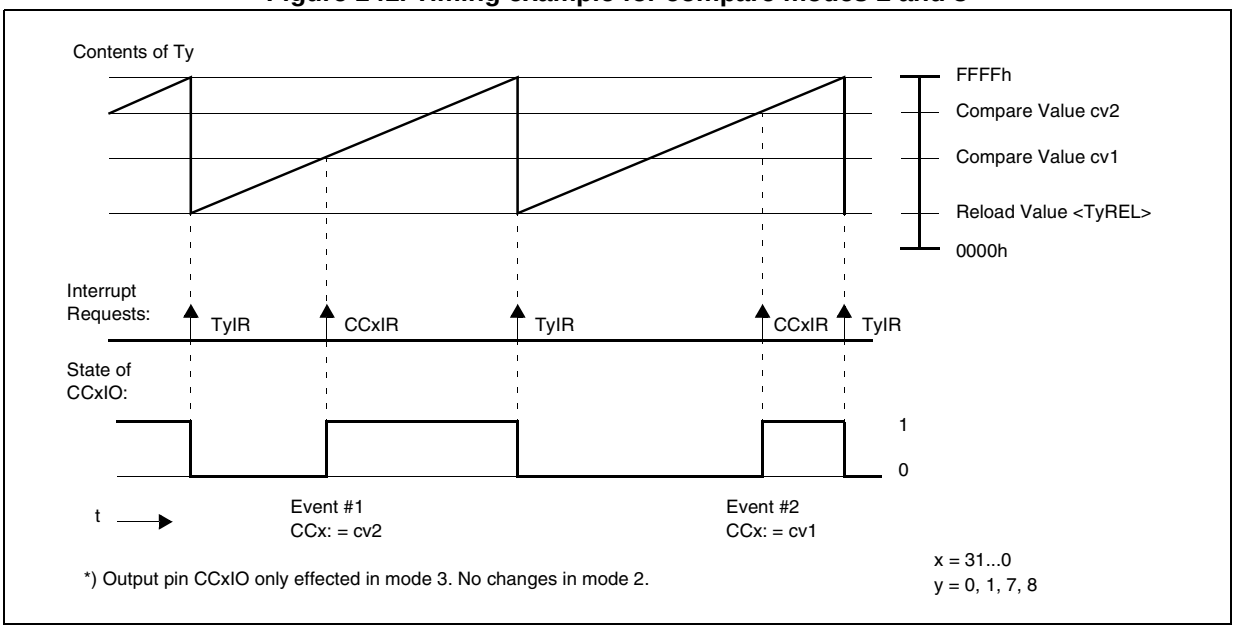
In *Figure 140*, the compare value in register CCx is modified from cv1 to cv2 after compare event #1. Compare event #2, however, will not occur until the next period of timer Ty.

Figure 140. Compare mode 2 and 3 block diagram



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

Figure 141. Timing example for compare modes 2 and 3



16.5.4 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 141*).

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.

16.5.5 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 50*.

Table 50. 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 50*) must be programmed to compare mode 1 and the corresponding bank 2 register (see *Table 50*) 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).

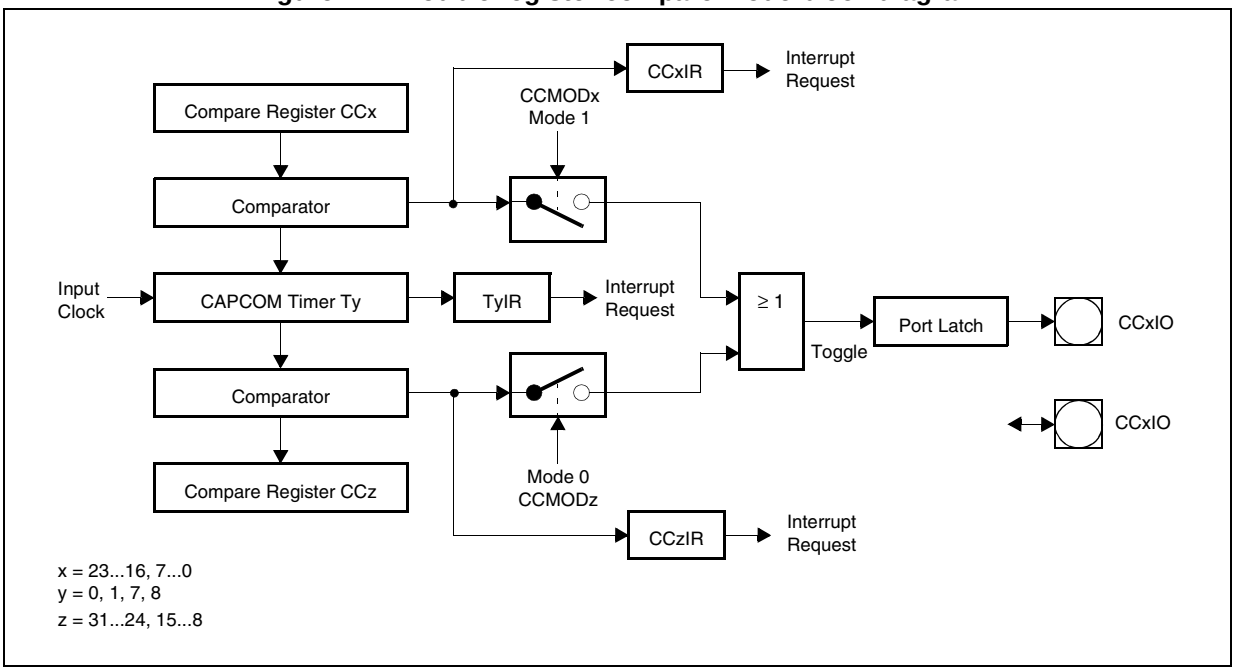
In the following, a bank 2 register (programmed to compare mode 0) will be referred to as CCz while the corresponding bank 1 register (programmed to compare mode 1) will be referred to as CCx.

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.

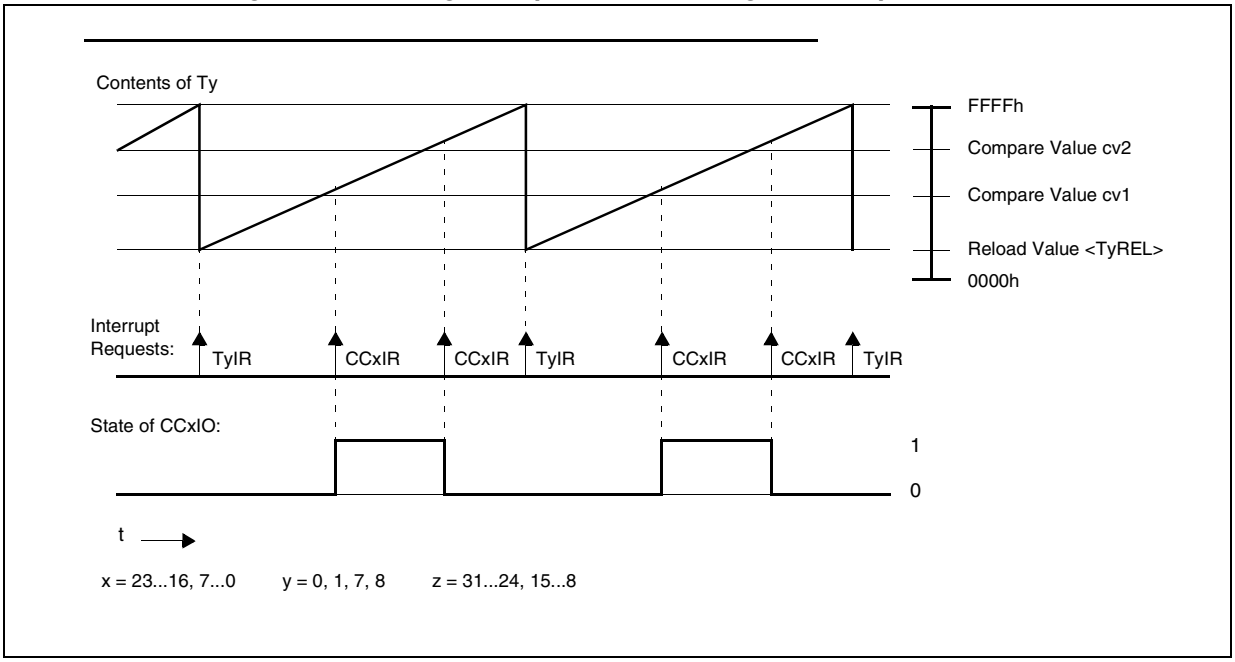
Figure 142. Double register compare mode block diagram



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. An example of this compare mode is shown in *Figure 143*. 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 143. Timing example for double register compare mode



16.6 Capture / compare interrupts

Upon a capture or compare event, the interrupt request flag CCxIR for the respective capture / compare register CCx is set to ‘1’. This flag can be used to generate an interrupt or trigger a PEC service request when enabled by the interrupt enable bit CCxIE.

Capture interrupts can be regarded as external interrupt requests with the additional feature of recording the time at which the triggering event occurred (see also section “External Interrupts”).

Note: Each of the 32 capture / compare registers (CC0...CC31) has its own bit-addressable interrupt control register (CC0IC...CC31IC) and its own interrupt vector (CC0INT...CC31INT). These registers are organized the same way as all other interrupt control registers. The figure below shows the basic register layout, and the table lists the associated addresses.

CCxIC (see *Table 51*) SFR/ESFR 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 Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

Table 51. 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

17 Pulse width modulation module

The Pulse Width Modulation (PWM) Module of the ST10F276 generates up to four independent PWM signals. The minimum PWM signal frequency depends on the width (16 bits) and the resolution (CLK/1 or CLK/64) of the PWM timers. The maximum PWM signal frequency assumes that the PWM output signal changes with every cycle of the respective timer. In a real application, the maximum PWM frequency will depend on the required resolution of the PWM output signal.

The pulse width modulation module has four 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 145*).

Figure 144. SFRs and port pins associated with the PWM module

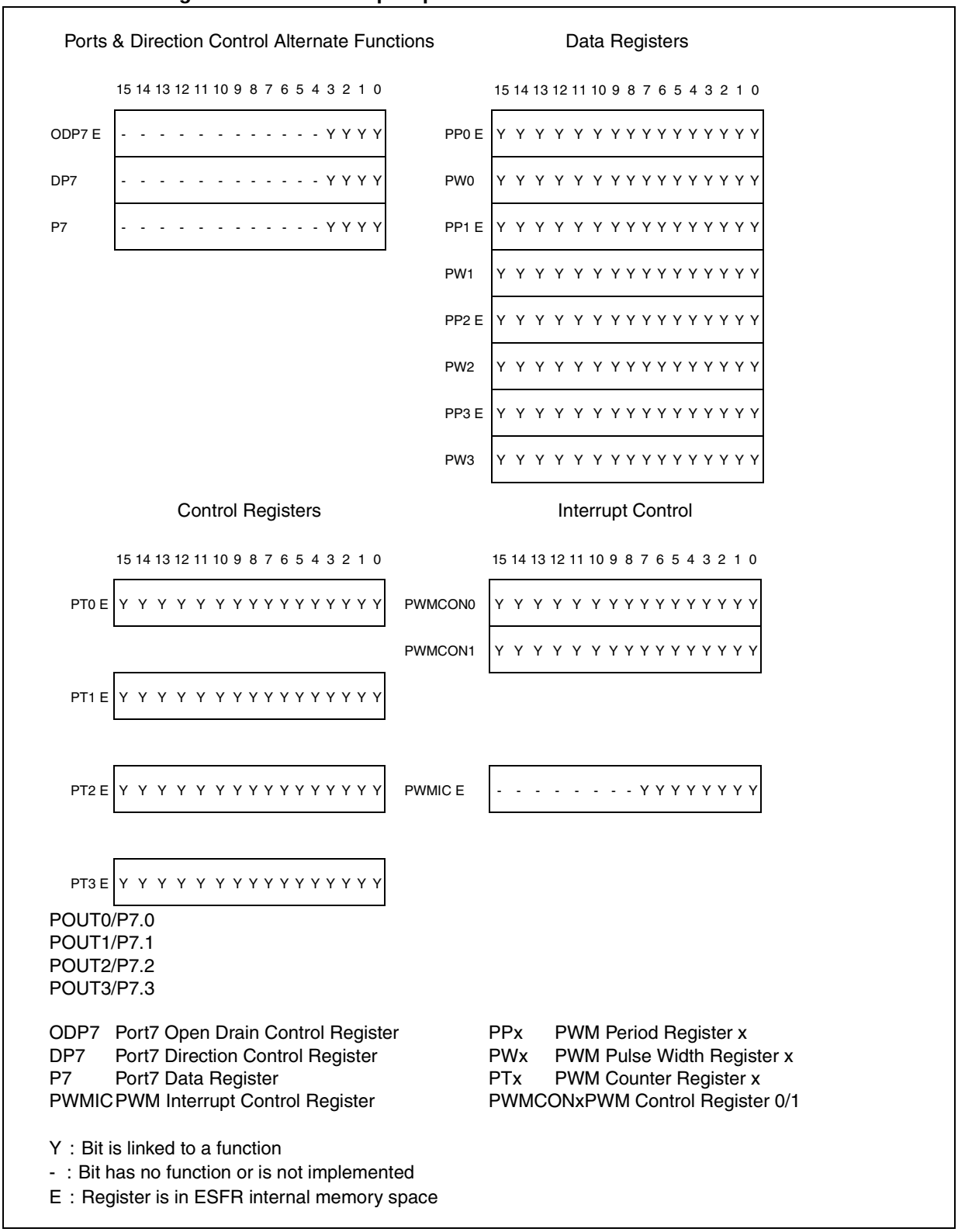
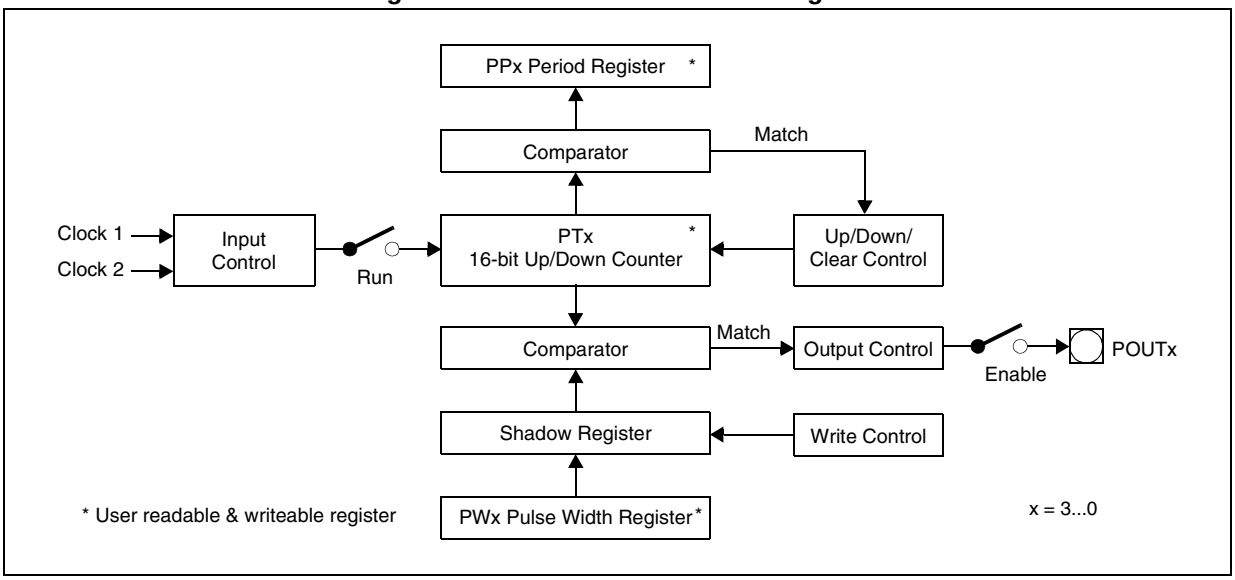


Figure 145. PWM channel block diagram



17.1 Operating modes

The PWM module provides four different operating modes:

- **Mode 0 standard PWM** generation (edge aligned PWM) available on four 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.

17.1.1 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.

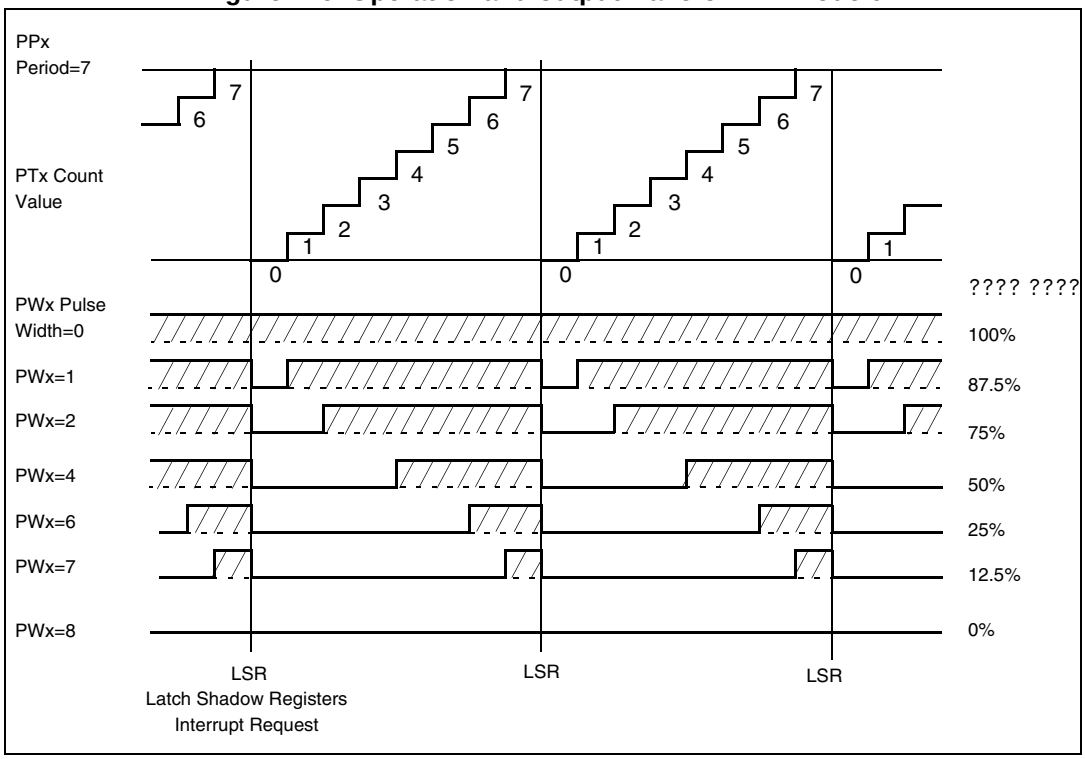
$$PWM_Period_{Mode0} = [PPx] + 1$$

The duty cycle of the PWM output signal is controlled by the value in the respective pulse width shadow register. This mechanism allows the selection of duty cycles from 0% to 100% including the boundaries.

For a value of 0000h the output will remain at a high level, representing a duty cycle of 100%. For a value higher than the value in the period register the output will remain at a low level, which corresponds to a duty cycle of 0%.

Figure 146 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 146. Operation and output waveform in mode 0



17.1.2 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.

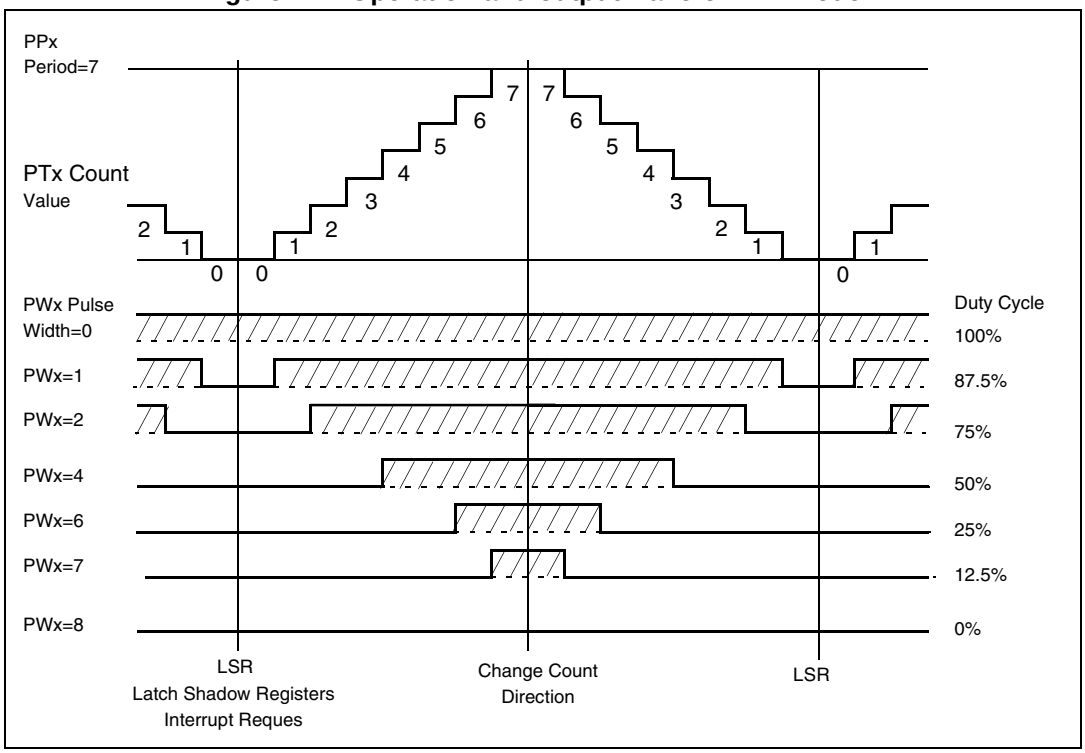
The PWM output signal is switched to a high level when the timer contents are equal to or greater than the contents of the pulse width shadow register while the timer is counting up. The signal is switched back to a low level when the respective timer has counted down to a value below the contents of 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:

$$PWM_Period_{Mode1} = 2 \times ([PPx] + 1)$$

Figure 147 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 147. Operation and output waveform in mode 1



17.1.3 Burst mode

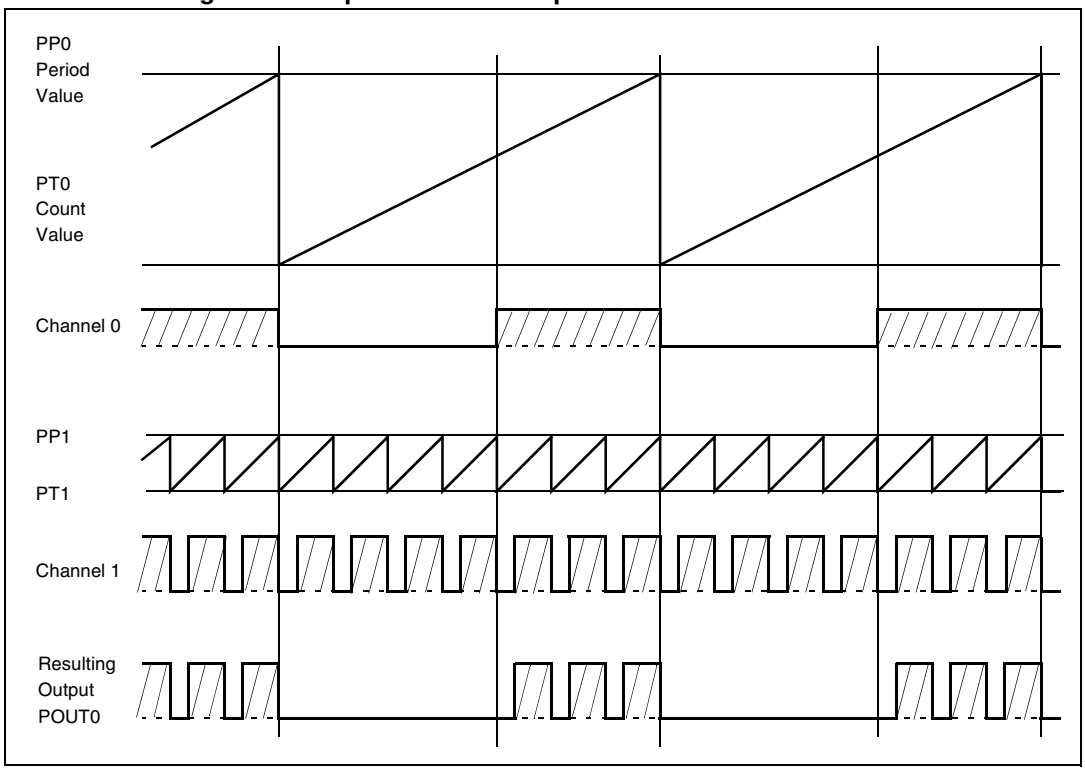
Burst mode is selected by setting bit PB01 in register PWMCON1 to '1'. This mode combines the signals from PWM channels 0 and 1 onto the port pin of channel 0.

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 148).

Figure 148. Operation and output waveform in burst mode



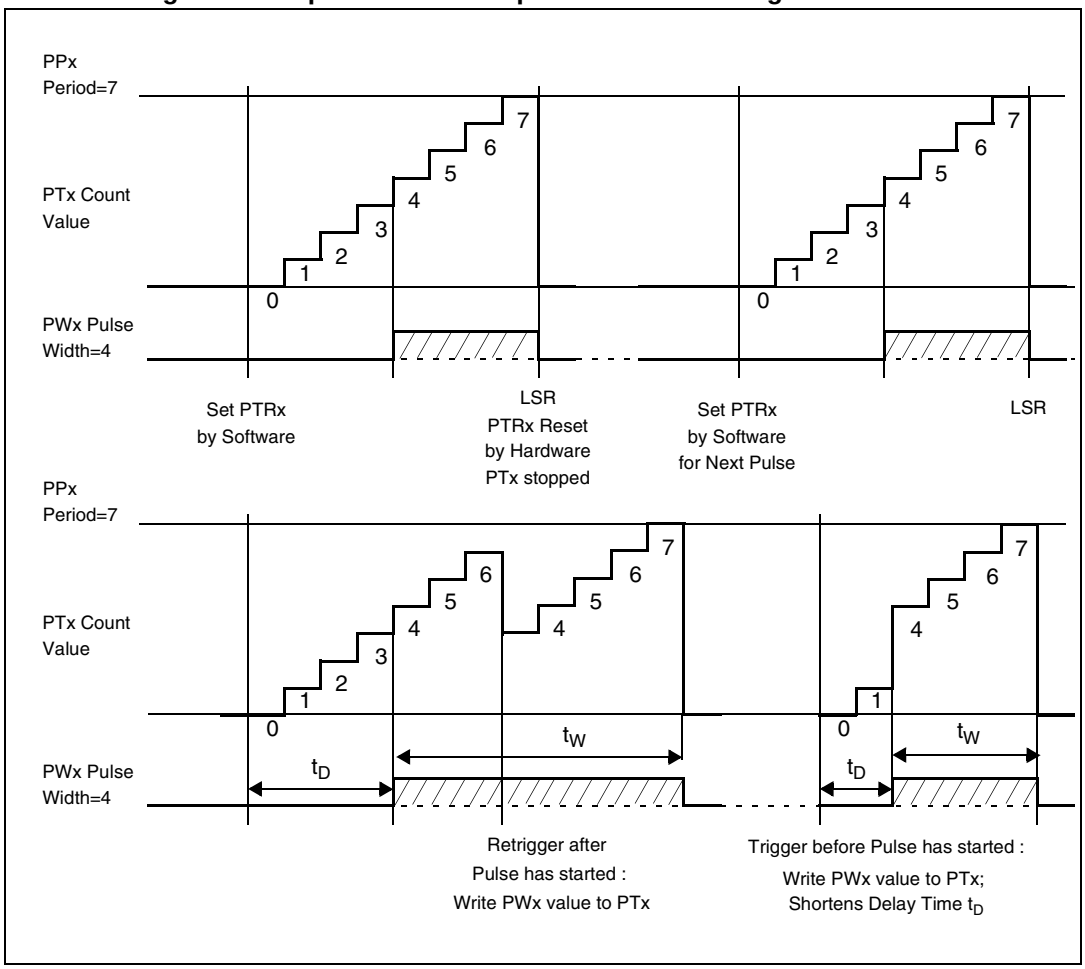
17.1.4 Single shot mode

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 149*).

Figure 149. 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 149*).

17.2 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.

Table 52 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 53*.

Table 52. PWM frequencies

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

Pulse width registers PWx

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:

PWM output signal = [PTx] $\geq$ [PWx shadow latch].

This type of comparison allows a flexible control of the PWM signal. For the register locations refer to the *Table 53*.

Table 53. 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 four PWM timers simultaneously with one bit-field instruction.

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

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

Bit	Function
PTRx	**PWM Timer x Run Control bit** '0': Timer PTx is disconnected from its input clock '1': Timer PTx is running
PTIx	**PWM Timer x Input Clock Selection** '0': Timer PTx clocked with CLK_{CPU} '1': Timer PTx clocked with CLK_{CPU} / 64
PIEx	**PWM Channel x Interrupt Enable Flag** '0': Interrupt from channel x disabled '1': Interrupt from channel x enabled
PIRx	**PWM Channel x Interrupt Request Flag** '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	**PWM Channel x Output Enable bit** ‘0’: Channel x output signal disabled, generate interrupt only ‘1’: Channel x output signal enabled
PMx	**PWM Channel x Mode Control bit** ‘0’: Channel x operates in mode 0, that is, edge aligned PWM ‘1’: Channel x operates in mode 1, that is, center aligned PWM
PB01	**PWM Channel 0/1 Burst Mode Control bit** ‘0’: Channels 0 and 1 work independently in respective standard mode ‘1’: Outputs of channels 0 and 1 are ANDed to POUT0 in burst mode
PSx	**PWM Channel x Single Shot Mode Control bit** ‘0’: Channel x works in respective standard mode ‘1’: Channel x operates in single shot mode

17.3 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 PIEx 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
-	-	-	-	-	-	-	-	PWMIR	PWMIE	ILVL				GLVL	
								RW	RW	RW				RW	

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for an explanation of the control fields.

17.4 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 150*).

Note: Using the open-drain mode on Port7 allows the combination of two or more PWM outputs through a AND-Wired 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:

PWM output signal = [PTx] $\geq$ [PWx shadow latch].

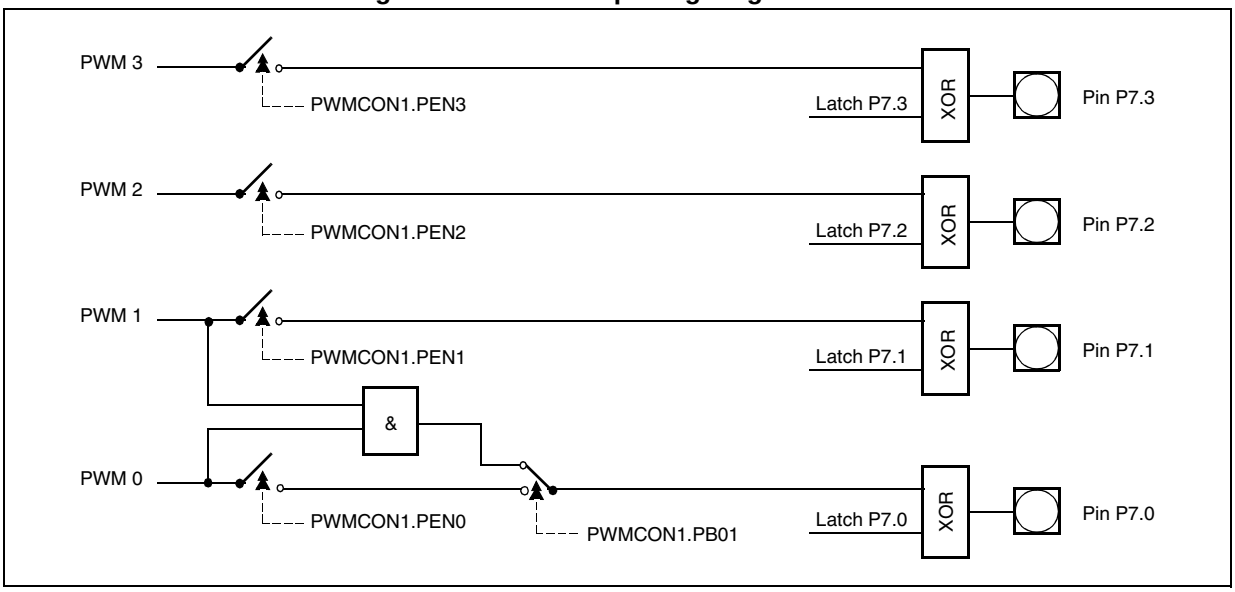
So whenever software changes registers PTx, the respective output will reflect the condition after the change. Loading timer PTx with a value greater than or equal to the value in PWx immediately sets the respective output, a PTx value below the PWx value clears the respective output.

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 150. PWM output signal generation



18 XBUS pulse width modulation module

A second Pulse Width Modulation (XPWM) Module is implemented on ST10F276. It is mapped on XBUS interface (Address range 00'EC00h-00'ECFFh) and generates up to four additional independent PWM signals. The minimum PWM signal frequency depends on the width (16 bits) and the resolution (CLK/1 or CLK/64) of the XPWM timers. The maximum PWM signal frequency assumes that the PWM output signal changes with every cycle of the respective timer. In a real application, the maximum PWM frequency will depend on the required resolution of the PWM output signal.

The pulse width modulation module has four independent PWM channels. Each channel has a 16-bit up/down counter XPTx, a 16-bit period register XPPx with a shadow latch, a 16-bit pulse width register XPWx with a shadow latch, two comparators, and the necessary control logic.

The main differences between PWM and XPWM are restricted to the programming model and interrupt management, due to the constraints imposed by the XBUS with respect to the standard ST10 peripheral bus (registers are not bit addressable, XBUS interrupt channels sharing with other X-Peripherals). In terms of general functionality and performance, the two modules are completely equivalent.

Figure 151. XBUS registers and port pins associated with the XPWM module

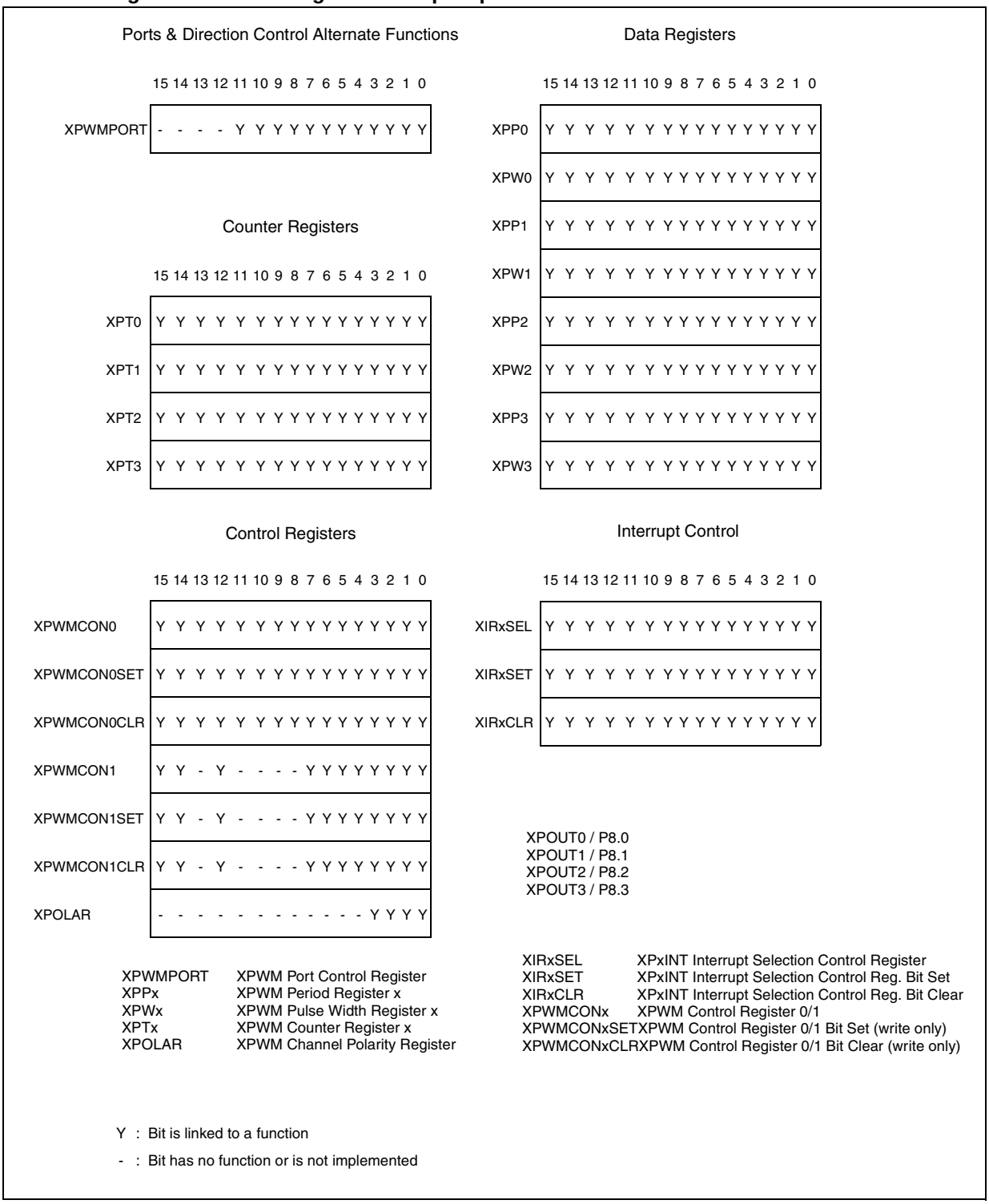
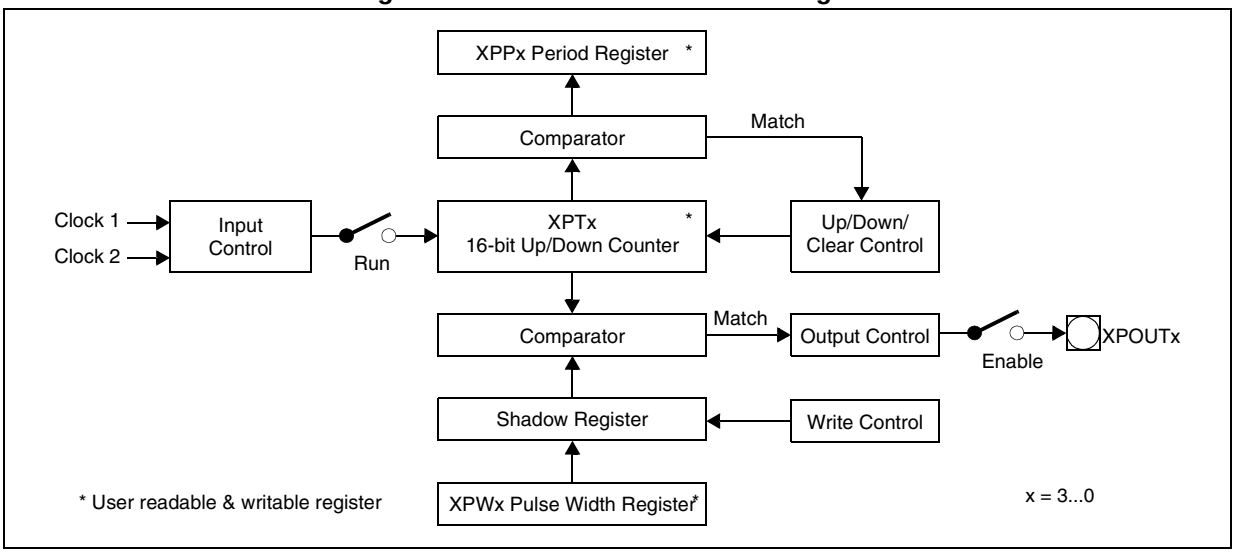


Figure 152. XPWM channel block diagram



18.1 Operating modes

The XPWM module provides four different operating modes:

- **Mode 0 standard PWM** generation (edge aligned PWM) available on four 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 XPWM 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. Besides, XPOLAR register is added to allow another level of possible polarity control. The descriptions below refer to the standard case after reset, which is direct drive.

18.1.1 Mode 0: standard PWM generation (edge aligned PWM)

Mode 0 is selected by clearing the respective bit PMx in register XPWMCON1 to '0'. In this mode the timer XPTx of the respective XPWM 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 XPWM 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 XPPx shadow register plus 1, counted in units of the timer resolution.

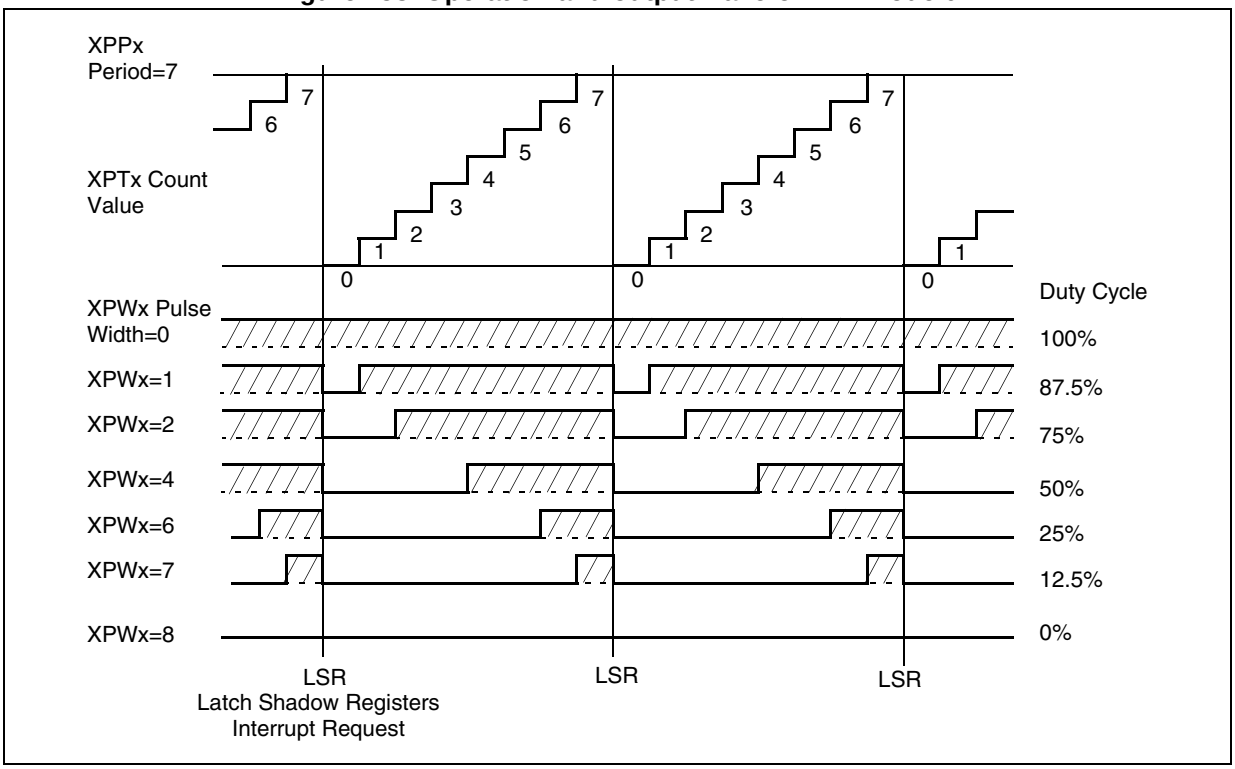
$$XPWM_Period_{Mode0} = [XPPx] + 1$$

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 153* illustrates the operation and output waveforms of a XPWM 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 153. Operation and output waveform in mode 0



18.1.2 Mode 1: symmetrical PWM generation (center aligned PWM)

Mode 1 is selected by setting the respective bit PMx in register XPWMCON1 to '1'. In this mode the timer XPTx of the respective XPWM 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.

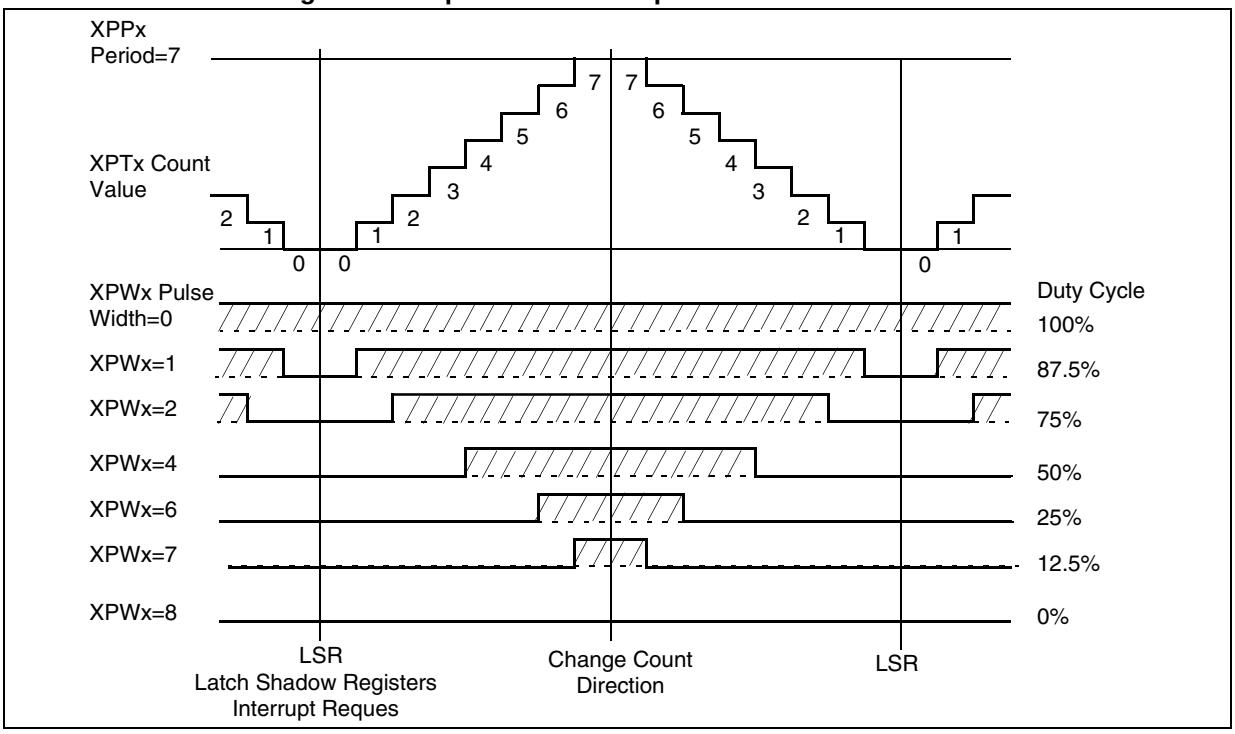
The PWM output signal is switched to a high level when the timer contents are equal to or greater than the contents of the pulse width shadow register while the timer is counting up. The signal is switched back to a low level when the respective timer has counted down to a value below the contents of 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:

$$XPWM_Period_{Mode1} = 2 \times ([XPPx] + 1)$$

The *Figure 147* illustrates the operation and output waveforms of a XPWM 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 154. Operation and output waveform in mode 1



18.1.3 Burst mode

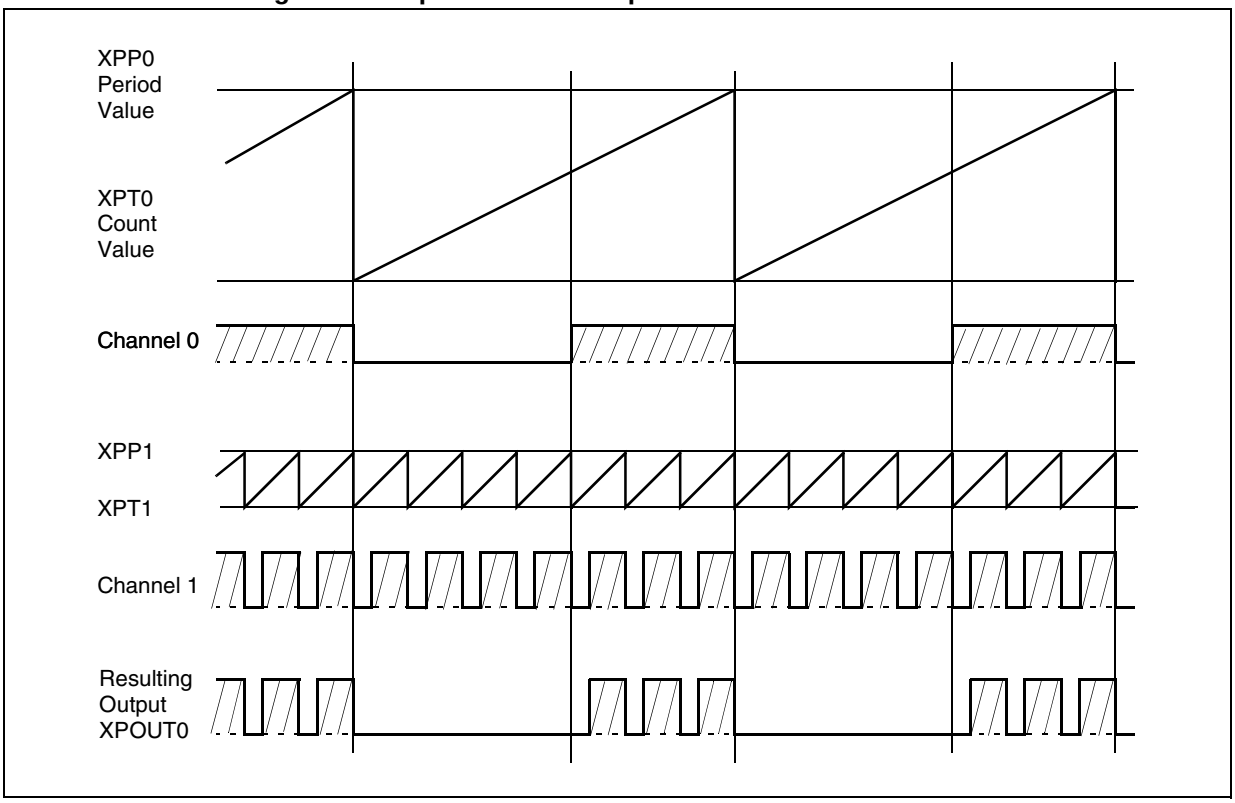
Burst mode is selected by setting bit PB01 in register XPWMCON1 to '1'. This mode combines the signals from XPWM channels 0 and 1 onto the port pin of channel 0.

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 148).

Figure 155. Operation and output waveform in burst mode



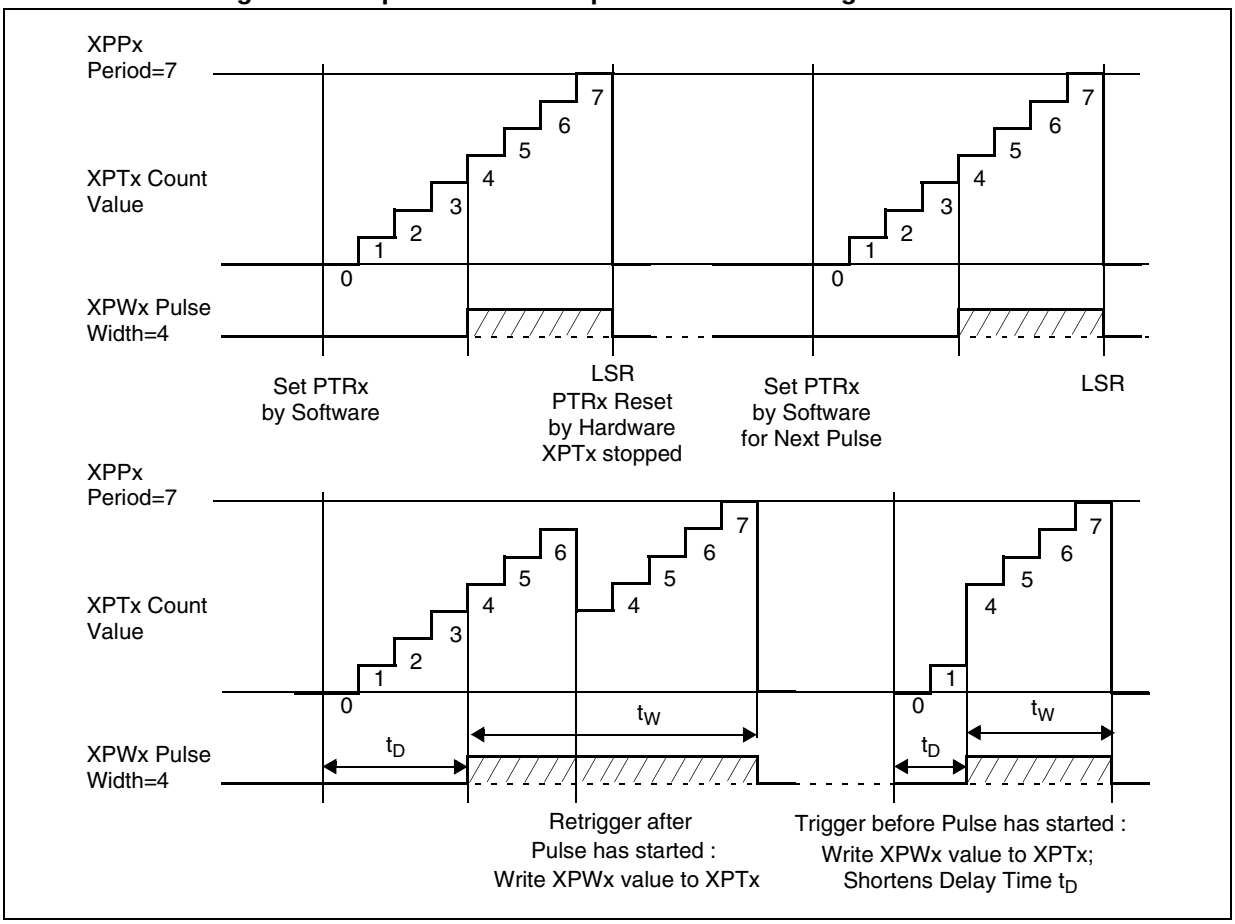
18.1.4 Single shot mode

Single shot mode is selected by setting the respective bit PSx in register XPWMCON1 to '1'. This mode is available for XPWM channels 2 and 3.

In this mode the timer XPTx of the respective XPWM 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 XPWM 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 149*).

Figure 156. 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 XPTx changes the positive and/or negative edge of the output signal, depending on whether the pulse has already started (the output is high) or not (the output is still low). This (multiple) re-triggering is always possible while the timer is running, after the pulse has started and before the timer is stopped.

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

By setting the period (XPPx), the timer start value (XPTx) and the pulse width value (XPWx) appropriately, the pulse width (t_W) and the optional pulse delay (t_D) may be varied in a wide range (see *Figure 149*).

18.2 XPWM module registers

The XPWM module is controlled via two sets of registers. The waveforms are selected by the channel specific registers XPTx (timer), XPPx (period) and XPWx (pulse width). Two common registers control the operating modes and the general functions (XPWMCON0 and XPWMCON1) of the XPWM module; the interrupt is controlled through the XBUS interrupt circuitry (X-Peripherals interrupt line sharing concept).

Up/down counters XPTx

Each counter XPTx of a XPWM channel is clocked either directly by the CPU clock or by the CPU clock divided by 64. Bit PTIx in register XPWMCON0 selects the respective clock source. A XPWM 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 XPTx rather than controlling the PWM output signal.

Table 52 summarizes the XPWM frequencies that result from various combinations of operating mode, counter resolution (input clock) and pulse width resolution.

Period registers XPPx

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

When a match is found between counter and XPPx shadow register, the counter is either reset to 0000h, or the count direction is switched from counting up to counting down, depending on the selected operating mode of that XPWM channel. For the register locations refer to *Table 53*.

Table 54. XPWM frequencies

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

Pulse width registers XPWx

This 16-bit register holds the actual PWM pulse width value which corresponds to the duty cycle of the PWM signal. This register is buffered with a shadow register.

The CPU accesses the XPWx register while the hardware compares the contents of the shadow register with the contents of the associated counter XPTx. The shadow register is loaded from the respective XPWx register at the beginning of every new PWM cycle, or upon a write access to XPWx, while the timer is stopped.

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

PWM output signal = [XPTx] $\geq$ [XPWx shadow latch]

This type of comparison allows a flexible control of the PWM signal. For the register locations refer to *Table 53*.

Table 55. XPWM module channel specific register addresses

Register	Address	Reg. space	Register	Address	Reg. space
XPW0	EC30h	XBUS	XPT0	EC10h	XBUS
XPW1	EC32h	XBUS	XPT1	EC12h	XBUS
XPW2	EC34h	XBUS	XPT2	EC14h	XBUS
XPW3	EC36h	XBUS	XPT3	EC16h	XBUS
All XPWM registers are not bit-addressable.			XPP0	EC20h	XBUS
			XPP1	EC22h	XBUS
			XPP2	EC24h	XBUS
			XPP3	EC26h	XBUS

XPWM control register XPWMCON0

Register XPWMCON0 controls the function of the timers of the four XPWM channels and the channel specific interrupts. Having the control bit organized in functional groups allows to start or to stop all the four XPWM timers simultaneously with one bit-field instruction.

Note: This register is not bit-addressable; the bit-addressability is available via specific 'set' and 'Clear' write-only registers XPWMCON0SET and XPWMCON0CLR.

XPWMCON0 (EC00h) XBUS Reset Value: 0000h

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

Bit	Function
PTRx	**XPWM Timer x Run Control bit** '0': Timer XPTx is disconnected from its input clock '1': Timer XPTx is running
PTIx	**XPWM Timer x Input Clock Selection** '0': Timer XPTx clocked with CLK_{CPU} '1': Timer XPTx clocked with CLK_{CPU} / 64
PIEx	**XPWM Channel x Interrupt Enable Flag** '0': Interrupt from channel x disabled '1': Interrupt from channel x enabled
PIRx	**XPWM Channel x Interrupt Request Flag** '0': No interrupt request from channel x '1': Channel x interrupt pending (must be reset via software)

XPWMCON0SET (EC06h) XBUS Reset Value: 0000h

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

Bit	Function
SET.Y	**XPWMCON0 Bit Y Set** Writing a '1' will set the corresponding bit in XPWMCON0 register. Writing a '0' has no effect.

XPWMCON0CLR (EC08h) XBUS Reset Value: 0000h

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

Bit	Function
CLR.Y	**XPWMCON0 Bit Y Clear** Writing a '1' will clear the corresponding bit in XPWMCON0 register, Writing a '0' has no effect.

XPWM control register XPWMCON1

Register XPWMCON1 controls the operating modes and the outputs of the four XPWM 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 XPWM channel is individually enabled by bit PENx. If the output is not enabled the respective pin can be used for general purpose I/O and the XPWM channel can only be used to generate an interrupt request.

Note: This register is not bit-addressable; the bit-addressability is available via specific 'set' and 'Clear' write-only registers XPWMCON1SET and XPWMCON1CLR.

XPWMCON1 (EC02h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS3	PS2	-	PB01	-	-	-	-	PM3	PM2	PM1	PM0	PEN3	PEN2	PEN1	PEN0
RW	RW		RW					RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
PENx	**XPWM Channel x Output Enable bit** '0': Channel x output signal disabled, generate interrupt only '1': Channel x output signal enabled
PMx	**XPWM Channel x Mode Control bit** '0': Channel x operates in mode 0, that is, edge aligned PWM '1': Channel x operates in mode 1, that is, center aligned PWM

Bit	Function
PB01	**XPWM Channel 0/1 Burst Mode Control bit** '0': Channels 0 and 1 work independently in respective standard mode '1': Outputs of channels 0 and 1 are ANDed to POUT0 in burst mode
PSx	**XPWM Channel x Single Shot Mode Control bit** '0': Channel x works in respective standard mode '1': Channel x operates in single shot mode

XPWMCON1SET (EC0Ah) XBUS Reset Value: 0000h

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

Bit	Function
SET.Y	**XPWMCON1 Bit Y Set** Writing a '1' will set the corresponding bit in XPWMCON1 register. Writing a '0' has no effect.

XPWMCON1CLR (EC0Ch) XBUS Reset Value: 0000h

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

Bit	Function
CLR.Y	**XPWMCON1 Bit Y Clear** Writing a '1' will clear the corresponding bit in XPWMCON1 register. Writing a '0' has no effect.

18.3 Interrupt request generation

Each of the four channels of the XPWM module can generate an individual interrupt request. Each of these "channel interrupts" can activate the common "module interrupt", which actually interrupts the CPU.

Up to four interrupt control registers (XIRxSEL, x = 0, 1, 2, 3) are provided in order to select the source of the XBUS interrupts: the XPWM common interrupt line, is linked to one of the XPxINT registers (x = 0, 1, 2, 3). In particular, the XPWM interrupt line is available on the XP2INT and XP3INT interrupt vectors.

Refer to *Section 5.7: X-peripheral interrupt on page 117* for details.

The interrupt service routine can determine the active channel interrupt(s) from the channel specific interrupt request flags PIRx in register XPWMCON0.

The interrupt request flag PIRx of a channel is set at the beginning of a new PWM cycle, when loading the shadow registers. This indicates that registers XPPx and XPWx are now ready to receive a new value. If a channel interrupt is enabled via its respective PIEx bit,

also the common interrupt request line is asserted (XP2IR/XP3IR in registers XP2IC/XP3IC), provided that they are enabled via the common interrupt enable bits (XP2IE/XP3IE) and the dedicated enable bits inside XIRSEL2/XIRSEL3 registers.

Note: The channel interrupt request flags (PIRx in register XPWMCON0) are not automatically cleared by hardware upon entry into the interrupt service routine, so they must be cleared via software. The module interrupt request flags XP2IR/XP3IR are also not 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.

18.4 XPWM output signals

The output signals of the four XPWM channels (XPOUT3...XPOUT0) are alternate output functions on Port8 (P8.3...P8.0). The XPWM signals are XORed with the outputs of the register XPOLAR before being driven to the port pins. This allows driving the XPWM signal directly to the port pin (XPOLAR.x = '0') or drive the inverted XPWM signal (XPOLAR.x = '1'). At Port8 level, when output direction is enabled, and bit XPWMEN of XPERCON is set, it is again possible to control the polarity like it is done for the standard PWM (on Port7), simply setting or clearing the output data register P8.x (x = 0...3); maintaining cleared the data register, the polarity is not inverted, while it is inverted setting the register. Note that if bit XP8.y (XPWMPORT register) is set, the polarity is inverted as well: so, if both XPOLAR.y and XP8.y are set, no inversion is achieved. For Port8 block diagram, refer also to *Section 6.10: Port8 on page 172*.

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 (P8.x = '0') or drive the inverted PWM signal (P8.x = '1') (see *Figure 150 on page 356*).

Note: Using the open-drain mode on Port8 allows the combination of two or more XPWM outputs through a AND-Wired configuration, using an external pull-up device. This provides sort of a burst mode for any XPWM channel.

XPOLAR (EC04h) XBUS Reset Value: 0000h

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

Bit	Function
XPOLAR.Y	**XPWM Channel Y Polarity Bit** '0': Polarity of Channel Y is normal '1': Polarity of Channel Y is inverted

Software control of the XPWM outputs

In an application the XPWM output signals are generally controlled by the XPWM module. However, it may be necessary to influence the level of the XPWM 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 XPWM channel outputs are controlled by comparators according to the formula:

PWM output signal = [XPTx] ≥ [XPWx shadow latch]

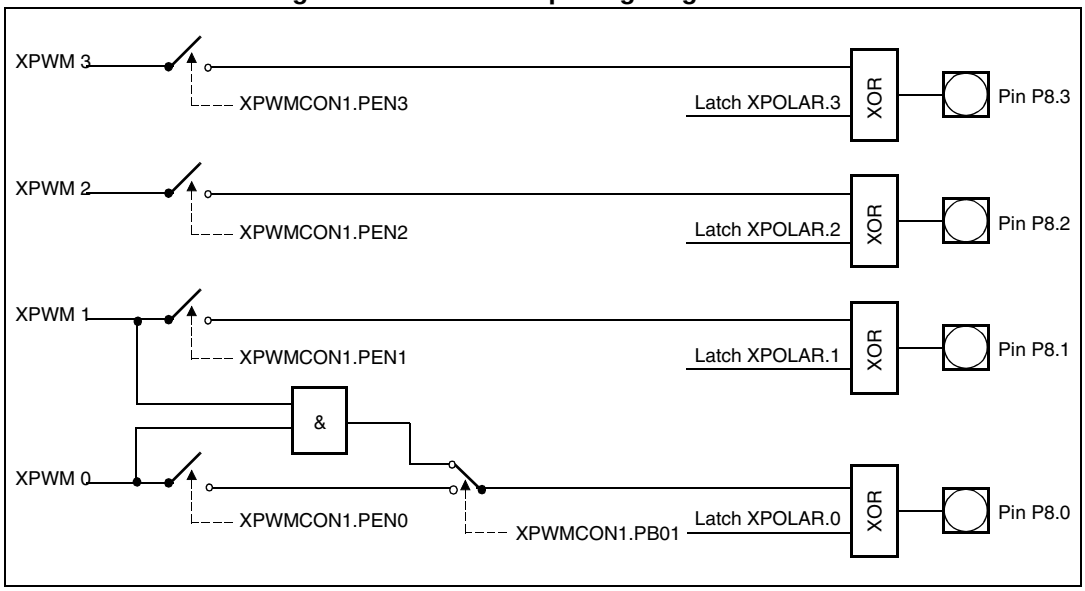
So whenever software changes registers XPTx, the respective output will reflect the condition after the change. Loading timer XPTx with a value greater than or equal to the value in XPWx immediately sets the respective output, a XPTx value below the XPWx value clears the respective output.

By clearing or setting the respective Port8 output latch the XPWM channel signal is driven directly or inverted to the port pin.

Clearing the enable bit PENx disconnects the XPWM 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 157. XPWM output signal generation



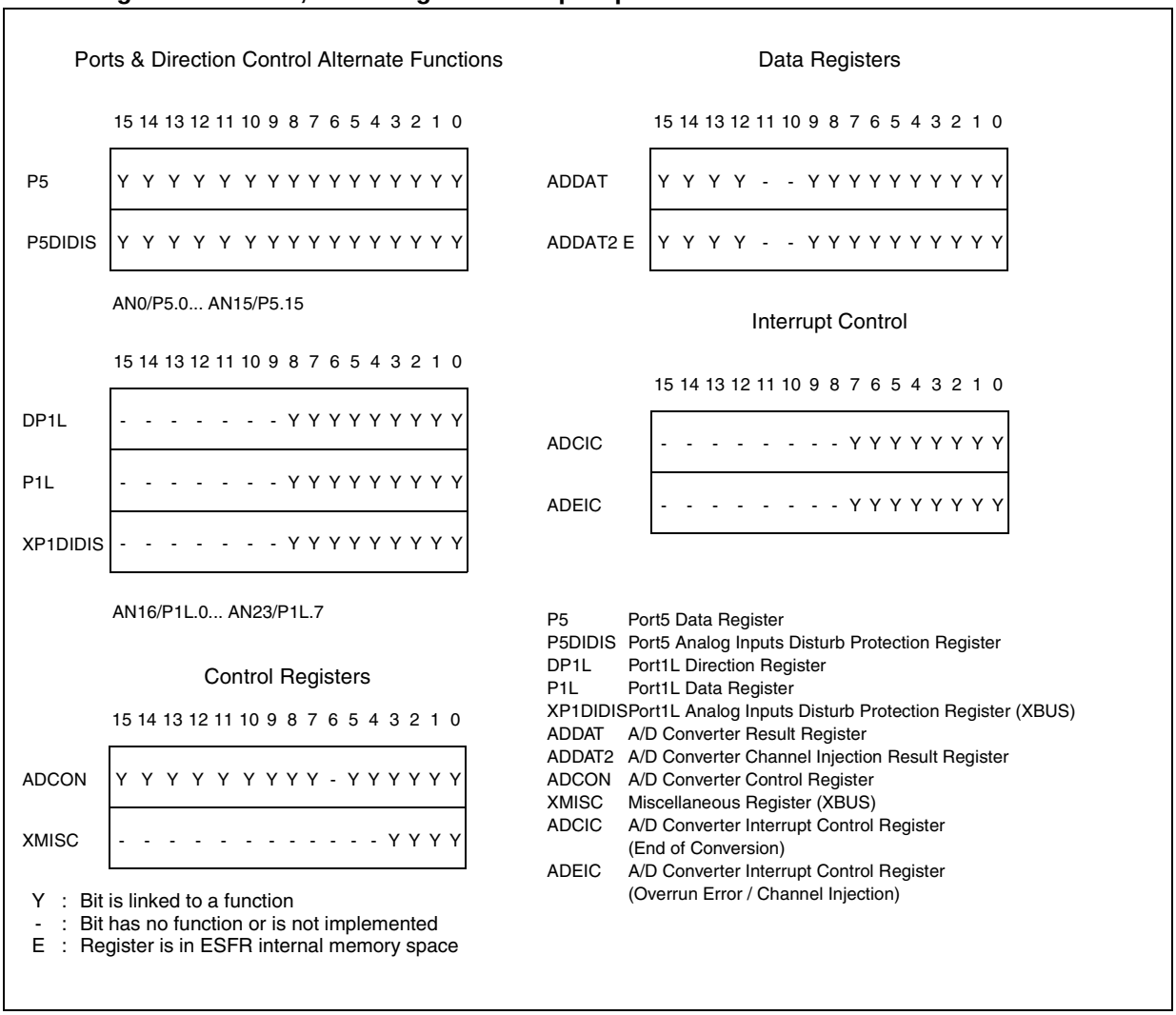
19 Analog / digital converter

The ST10F276 provides an analog / digital converter with 10-bit resolution and a sample & hold circuit on-chip. A multiplexer selects between up to 16+8 analog input channels (alternate functions of Port5 and Port1) either via software (fixed channel modes) or automatically (auto scan modes works automatically among the 16 channels of Port5, or among the 8 channels of Port1). An automatic self-calibration adjusts the ADC module to process parameter variations at each reset event. 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.

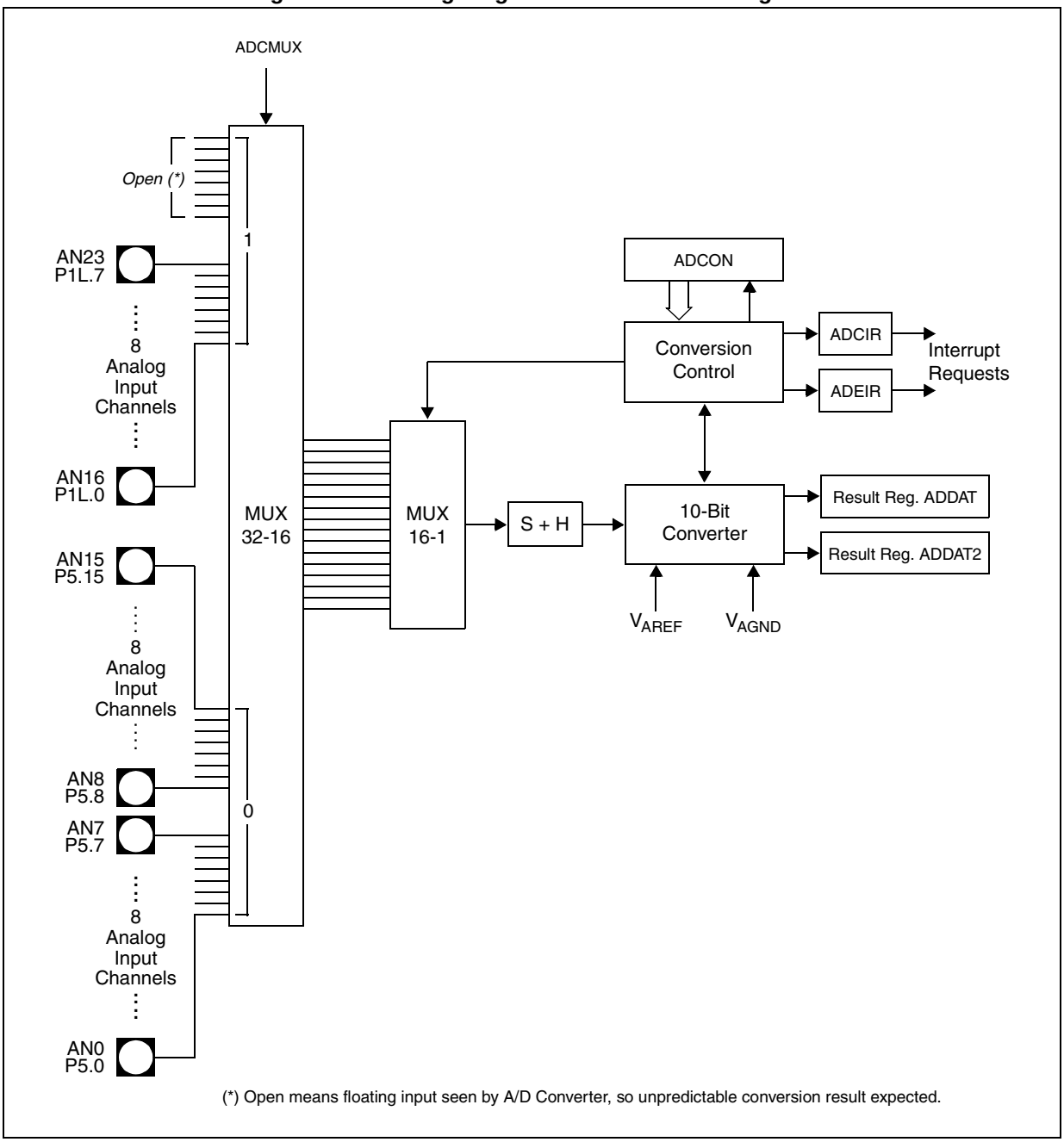
Figure 158. SFRs, XBUS registers and port pins associated with the A/D converter



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 159. Analog / digital converter block diagram



19.1 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 additional register P5DIDIS can be used to further protect ADC input analog section disabling the digital input section. *Section 6.7.2: Port5 analog inputs disturb protection on page 163* for details on register P5DIDIS.

The analog input channels AN16...AN23 are alternate functions of Port1L which is an 8-bit bidirectional port. The Port1L lines may either be used as analog input or digital input/output. The additional register XP1DIDIS can be used to further protect ADC input analog section disabling the digital input section. Refer to *Section 6.3.2: PORT1 analog inputs disturb protection on page 144* for details on register XP1DIDIS.

In order to configure the Port1L lines as analog inputs it is first of all recommended to properly set register XP1DIDIS; next it is necessary to set bit ADCMUX of register XMISC: in this way all analog input channels of Port5 are disabled, and the analog signal to the Converter is provided through the Port1L pins.

Both XMISC and XP1DIDIS registers can be accessed only after bit XMISCEN of register XPERCON and bit XPEN of register SYSCON have been set.

XMISC (EB46h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	VREG OFF	CAN CK2	CAN PAR	ADC MUX
												RW	RW	RW	RW

Bit	Function
ADCMUX	**Port1L ADC Channels Enable** ‘0’: Analog inputs on port P5.y can be converted (default configuration) ‘1’: Analog inputs on port P1.z can be converted. Only 8 channels can be managed
CANPAR	**CAN Parallel Mode Selection** ‘0’: CAN2 is mapped on P4.4/P4.7, while CAN1 is mapped on P4.5/P4.6 ‘1’: CAN1 and CAN2 are mapped in parallel on P4.5/P4.6. This is effective only if both CAN1 and CAN2 are enabled through setting of bits CAN1EN and CAN2EN in XPERCON register. If CAN1 is disabled, CAN2 remains on P4.4/P4.7 even if bit CANPAR is set.
CANCK2	**CAN Clock divider by 2 disable** ‘0’: Clock provided to CAN modules is CPU clock divided by 2 (mandatory when f_{CPU} is higher than 40 MHz) ‘1’: Clock provided to CAN modules is directly CPU clock
VREGOFF	**Main Voltage Regulator disable for Power Down mode** ‘0’: Default value after reset and when Power Down is not used. ‘1’: On-chip Main Regulator is turned off when Power Down mode is entered.

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		AD CRQ	AD CIN	AD WR	AD BSY	ADST	AD OFF	ADM		ADCH			
RW		RW		RW	RW	RW	R	RW	RW	RW		RW			

Bit	Function
ADCH	ADC Analog Channel Input Selection
ADM	**ADC Mode Selection** '0 0': Fixed Channel Single Conversion '0 1': Fixed Channel Continuous Conversion '1 0': Auto Scan Single Conversion '1 1': Auto Scan Continuous Conversion
ADOFF	**ADC Disable** '0': Analog circuitry of A/D converter is on: it can be used properly '1': Analog circuitry of A/D converter is turned off (no consumption): non conversion possible
ADST	ADC Start bit
ADBSY	**ADC Busy Flag** '1': a conversion or calibration is active
ADWR	ADC Wait for Read Control
ADCIN	ADC Channel Injection Enable
ADCRQ	ADC Channel Injection Request Flag
ADSTC	ADC Sample Time Control (1)
ADCTC	ADC Conversion Time Control (1)

1. ADSTC and ADCTC control the conversion timing. Refer to Section 20.2.

Bit-field ADCH specifies the analog input channel which is to be converted (first channel of a conversion sequence in auto scan modes): according to the value of bit ADCMUX in XMISC register, a channel on Port5 or Port1L is selected. 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. After reset, this bit is set because the self-calibration is on-going (duration of self-calibration depends on CPU clock: it takes up to 40.629 ± 1 clock pulses). You must poll this bit to know when the first conversion can be launched.

The result of a conversion is stored in the result register ADDAT, or in register ADDAT2 for an injected conversion.

Note: Bit-field CHNR of register ADDAT is loaded by the ADC to indicate which channel the result refers to. Bit-field CHNR of register ADDAT2 is loaded by the CPU to select the analog channel, which is to be injected.

ADDAT (FEA0h / 50h) SFR Reset Value: 0000h

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

ADDAT2 (F0A0h / 50h) ESFR Reset Value: 0000h

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

Bit	Function
ADRES	A/D Conversion Result (10 bits)
CHNR	**Channel Number** 4 bits, identifies the converted analog channel: first eight binary values should be combined with status of bit ADCMUX of register XMISC to distinguish between channel(7:0) and channel(23:16).

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.

Field ADCH represents the channel of Port5 when bit ADCMUX in register XMISC is not set (0h = channel 0, 1h = channel 1, ... , Fh = channel 15); while it represents the channel of Port1 when ADCMUX is set (0h = channel 16, 1h = channel 17, ... , 7h = channel 23).

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).

19.1.1 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. If bit ADCMUX of register XMISC is set, the converted channel is the one on Port1 (ADCH = 0h for channel 16, ADCH = 1h for channel 17, ... , ADCH = 7h for channel 23). 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.

19.1.2 Auto scan conversion modes

These modes are selected by programming the mode selection field ADM in register ADCON to '10_B' (single conversion) or to '11_B' (continuous conversion).

Auto Scan modes automatically convert a sequence of analog channels, beginning with the channel specified in bit-field ADCH and ending with channel 0, without requiring software to change the channel number. Again, if bit ADCMUX of register XMISC is set, the sequence starts with the specified channel on Port1 ending with channel 16. Besides, if bit ADCMUX of register XMISC is set and ADCH value is greater than 7h, the sequence starts converting non existing channel: this corresponds to an unpredictable result, since the input of A/D Converter is left floating. After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bit-field ADCH will be converted.

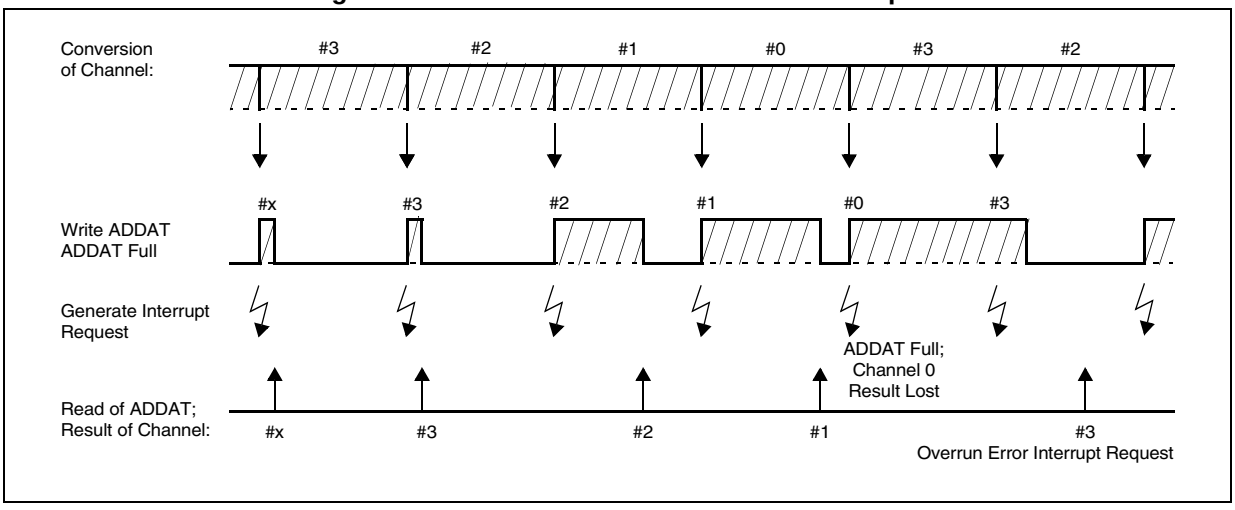
After the conversion is complete, the interrupt request flag ADCIR will be set and the converter will automatically start a new conversion of the next lower channel. ADCIR will be set after each completed conversion. After conversion of channel 0 the current sequence is complete.

In single conversion mode the converter will automatically stop and reset bits ADBSY and ADST.

In continuous conversion mode the converter will automatically start a new sequence beginning with the conversion of the channel specified in ADCH.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current sequence (including conversion of channel 0) and then stop and reset bit ADBSY.

Figure 160. Auto scan conversion mode example



19.1.3 Wait for ADDAT read mode

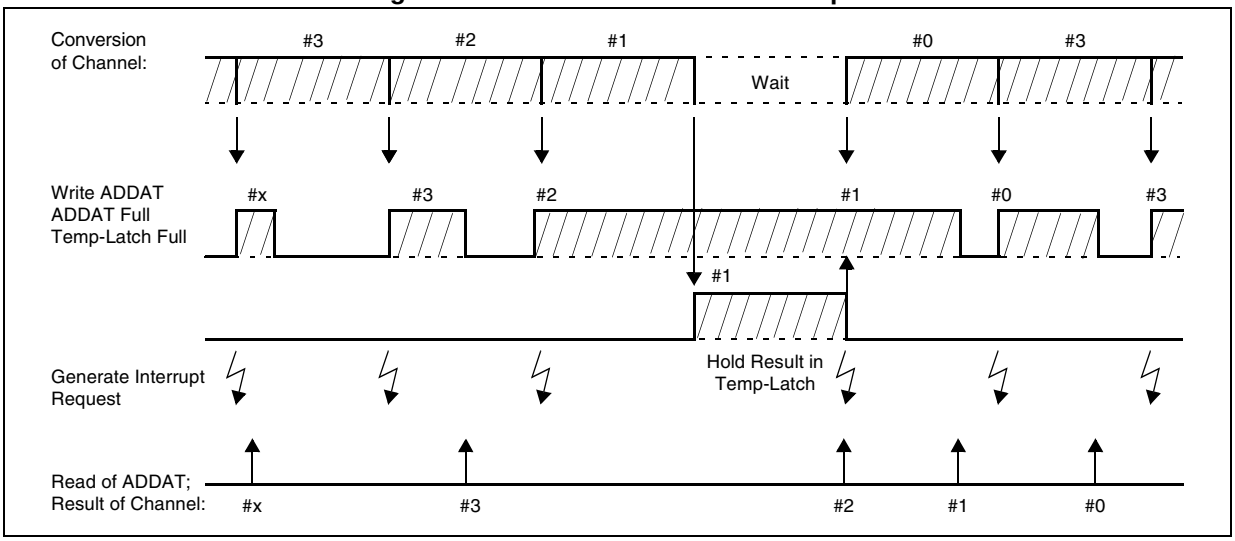
If in default mode of the ADC a previous conversion result has not been read out of register ADDAT by the time a new conversion is complete, the previous result in register ADDAT is lost because it is overwritten by the new value, and the A/D overrun error interrupt request flag ADEIR will be set.

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 161. Wait for read mode example



19.1.4 Channel injection mode

Channel Injection Mode allows the conversion of a specific analog channel (also while the ADC is running in a continuous or auto scan mode) without changing the current operating mode. After the conversion of this specific channel, the ADC continues with the original operating mode.

Channel Injection mode is enabled by setting bit ADCIN in register ADCON and requires the Wait for ADDAT Read Mode (ADWR = ‘1’). The channel to be converted in this mode is specified in bit-field CHNR of register ADDAT2.

These 4 bits in ADDAT2 are not modified by the A/D converter, but only the ADRES bit-field. Since the channel number for an injected conversion is not buffered, bit-field CHNR of ADDAT2 must never be modified during the sample phase of an injected conversion,

otherwise the input multiplexer will switch to the new channel. It is recommended to only change the channel number with no injected conversion running (see *Figure 162*).

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.

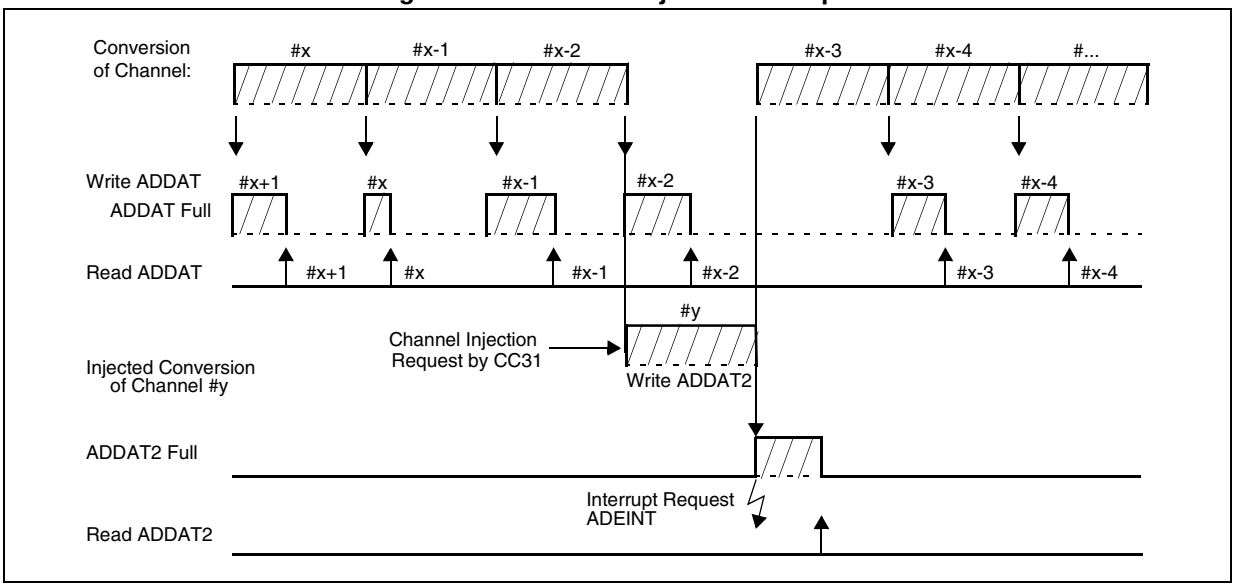
When Port1 channels are used in addition to those of Port5, attention must be paid in managing the channel injection mode. When the injection is controlled via software, the status of bit ADCMUX in register XMISC should be considered:

- If the channel to inject is on Port5, and ADCMUX is set, before setting bit ADCRQ in register ADCON, bit ADCMUX needs to be reset;
- If the channel to inject is on Port1, and ADCMUX is reset, before setting bit ADCRQ in register ADCON, bit ADCMUX needs to be set;
- If the channel to inject is on Port5, and ADCMUX is reset, setting bit ADCRQ in register ADCON will properly inject the right channel;
- If the channel to inject is on Port1, and ADCMUX is set, setting bit ADCRQ in register ADCON will properly inject the right channel.

When the injection is not controlled via software, the status of the bit ADCMUX cannot be checked: this implies that the analog signal to convert as injected channel, must be available on both the channel banks on Port5 and Port1. This is true when the channel to inject is mapped on P5(7:0) or P1(7:0): supposing channel to inject is channel #3, P5.3 and P1.3 pins must be externally shorted, providing the analog signal to both. On the contrary, when the channel to inject is on P5(15:8), there is no correspondent channel on Port1: this means that if ADCMUX is set at the time the injection is triggered, an unpredictable result is expected, since the A/D converter analog input is left floating.

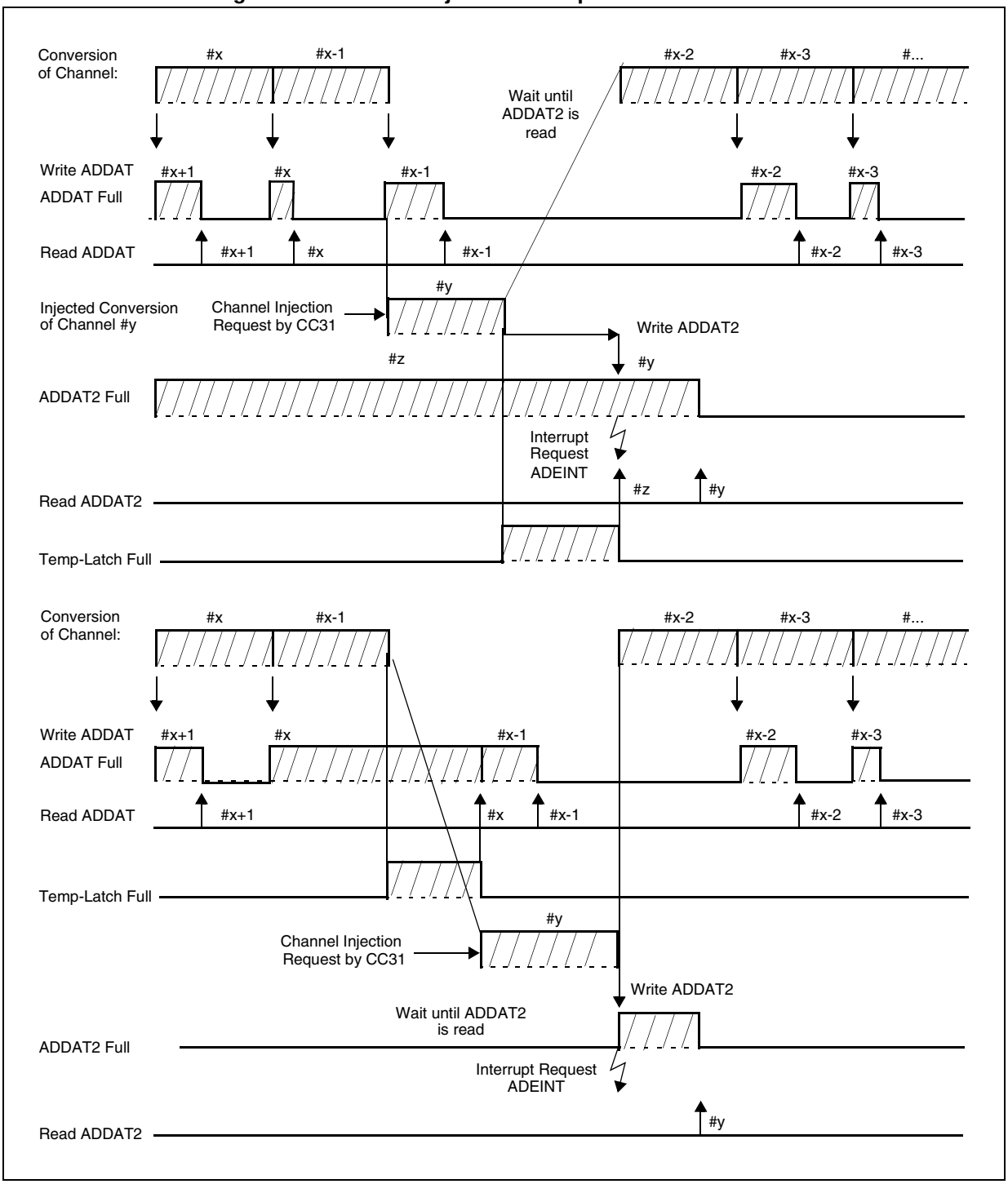
After the completion of the current conversion (if any is in progress) the converter will start (inject) the conversion of the specified channel. When the conversion of this channel is complete, the result will be placed into the alternate result register ADDAT2, and a Channel Injection Complete Interrupt request will be generated, which uses the interrupt request flag ADEIR (for this reason the Wait for ADDAT Read Mode is required).

Figure 162. 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 163. Channel injection example with wait for read



19.1.5 ADC power off (ADOFF)

Setting bit ADOFF in ADCON register the ADC is turned off and the static power consumption related with ADC analog circuitry is zeroed. If this bit is set during a conversion, the command is ignored (even though the ADOFF bit is immediately set): only

at the end of the conversion (or sequence of conversions if Scan mode was selected), the ADC is switched off (as soon as ADBSY bit is cleared).

When ADC is off (ADOFF bit set), setting bit ADST wakes automatically up the ADC and a conversion starts: the accuracy is unfortunately not yet granted, since the analog circuitry needs at least 50µs to complete the power-up transient phase. It is recommended to clear ADOFF bit first, and only after 50µs to start the first conversion.

Note: If bit ADOFF is set and when ADST is set also, at the end of the conversion (or cycle of conversion if Scan mode is selected), the ADC is switched off again (as soon as ADBSY is cleared).

Turning off ADC consumption (setting bit ADOFF) should be done once the Calibration is completed (starts after every reset occurrence): on the contrary, the calibration is stopped by setting bit ADOFF, and not restarted/completed once bit ADOFF is cleared again.

19.2 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 several successive steps, which correspond to the 10-bit resolution of the ADC. During these 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 ST10F276 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 ST10F276 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 bits of register ADCON. Bit fields ADCTC and ADSTC are used to define the basic conversion time and in particular the partition between sample phase and comparison phases. The table below lists the possible combinations. The timings refer to the unit TCL, where f_{CPU} = 1/2TCL.

Table 56. ADC sampling and conversion timing

ADCTC	ADSTC	Sample	Comparison	Extra	Total conversion
00	00	TCL * 120	TCL * 240	TCL * 28	TCL * 388
00	01	TCL * 140	TCL * 280	TCL * 16	TCL * 436
00	10	TCL * 200	TCL * 280	TCL * 52	TCL * 532
00	11	TCL * 400	TCL * 280	TCL * 44	TCL * 724
11	00	TCL * 240	TCL * 480	TCL * 52	TCL * 772
11	01	TCL * 280	TCL * 560	TCL * 28	TCL * 868
11	10	TCL * 400	TCL * 560	TCL * 100	TCL * 1060
11	11	TCL * 800	TCL * 560	TCL * 52	TCL * 1444
10	00	TCL * 480	TCL * 960	TCL * 100	TCL * 1540
10	01	TCL * 560	TCL * 1120	TCL * 52	TCL * 1732
10	10	TCL * 800	TCL * 1120	TCL * 196	TCL * 2116
10	11	TCL * 1600	TCL * 1120	TCL * 164	TCL * 2884

A complete conversion time includes the conversion itself, the sample time and the time required to transfer the digital value to the result register.

Note: The total conversion time is compatible with the formula valid for ST10F269, while the meaning of the bit fields ADCTC and ADSTC is no longer compatible: the minimum conversion time is 388 TCL, which at 40 MHz CPU frequency corresponds to 4.85µs (see ST10F269).
ST10F276 can target a maximum CPU frequency of 64 MHz. This means that the minimum conversion time is around 3µs.

19.3 A/D converter interrupt control

At the end of each conversion, interrupt request flag ADCIR in interrupt control register ADCIC is set. This end-of-conversion interrupt request may cause an interrupt to vector ADCINT, or it may trigger a PEC data transfer which reads the conversion result from register ADDAT it can be stored it into a table in the on-chip 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
-	-	-	-	-	-	-	-	ADEIR	ADEIE	ILVL				GLVL	
								RW	RW	RW				RW	

Note: Refer to Section 5.1.3: Interrupt control registers on page 100 for explanation of the control fields.

19.4 Calibration

A full calibration sequence is performed after a reset. This full calibration lasts 40.629 ± 1 CPU clock cycles. During this time, the busy flag ADBSY is set to indicate the operation. It compensates the capacitance mismatch, so the calibration procedure does not need any update during normal operation.

No conversion can be performed during this time: the bit ADBSY should be polled to verify when the calibration is over, and the module is able to start a conversion.

Since the calibration process writes repeatedly spurious conversion results inside ADDAT register, at the end of the calibration, both ADCIR and ADEIR flags are set. For this reason, before starting a conversion, in the A/D Converter initialization routine, the application should perform a dummy read of ADDAT register and clear the two flags.

Note: If ADDAT is not read before starting the first conversion, and for example "Wait for Read Mode" is entered (ADWR bit set), the A/D Converter is stack waiting for the ADDAT read, since the result of the current conversion cannot be immediately written inside ADDAT, which contains the results of the calibration (meaningless data).

19.5 A/D conversion accuracy

The A/D Converter compares the analog voltage sampled on the selected analog input channel to its analog reference voltage (V_{AREF}) and converts it into 10-bit digital data. The absolute accuracy of the A/D conversion is the deviation between the input analog value and the output digital value. It includes the following errors:

- Offset error (OFS)
- Gain Error (GE)
- Quantization error
- Non-Linearity error (Differential and Integral)

These four error quantities are explained below using *Figure 164*.

Offset Error

Offset error is the deviation between actual and ideal A/D conversion characteristics when the digital output value changes from the minimum (zero voltage) 00 to 01 (*Figure 164*, see OFS).

Gain Error

Gain error is the deviation between the actual and ideal A/D conversion characteristics when the digital output value changes from the 3FE to the maximum 3FF, once offset error is subtracted.

Gain error combined with offset error, represents the so-called full-scale error (*Figure 164*, OFS + GE).

Quantization error

Quantization error is the intrinsic error of the A/D converter and is expressed as 1/2 LSB.

Non-linearity error

Non-Linearity error is the deviation between actual and the best-fitting A/D conversion characteristics (see *Figure 164*):

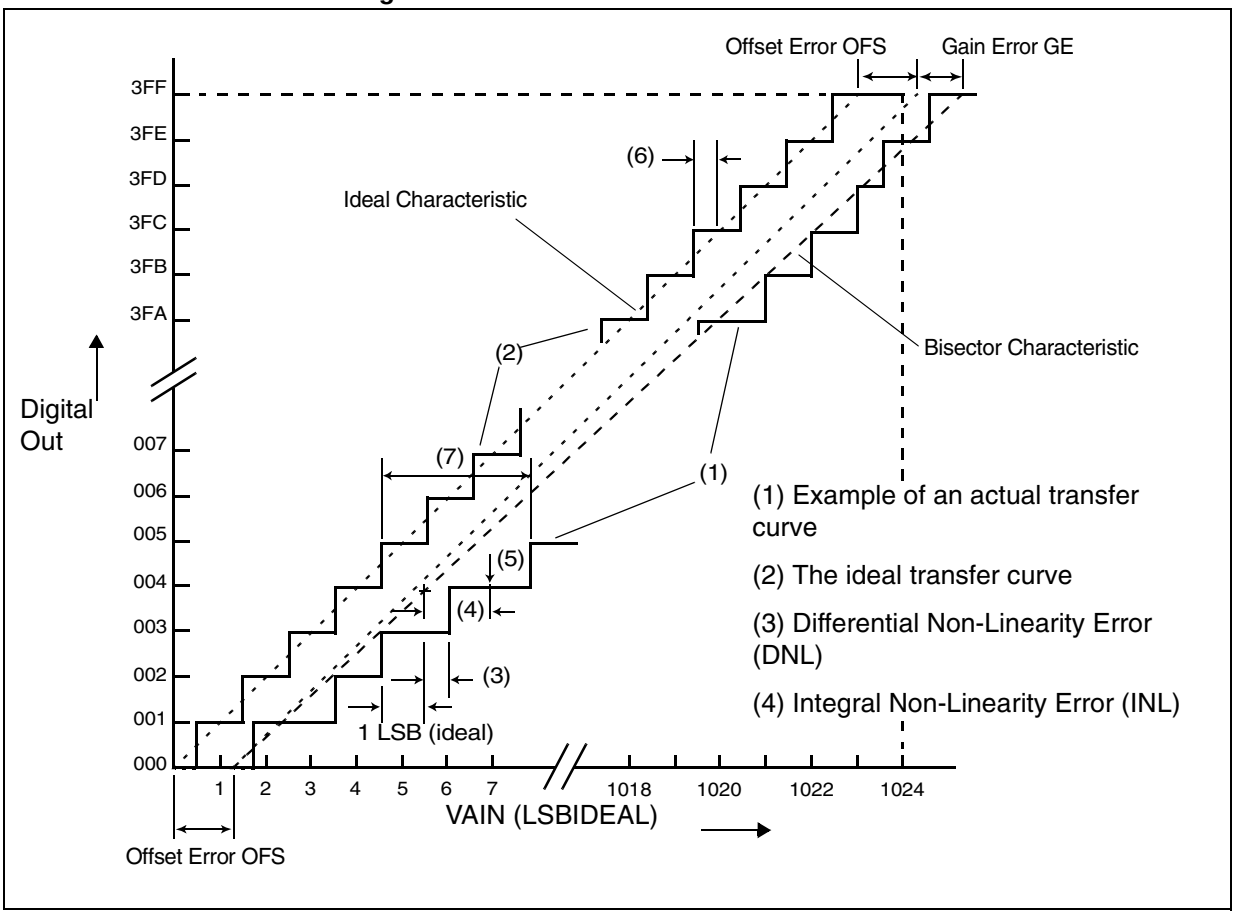
- Differential Non-Linearity error is the actual step dimension versus the ideal one (1 LSB_{IDEAL}).
- Integral Non-Linearity error is the distance between the center of the actual step and the center of the bisector line, in the actual characteristics. Note that for Integral Non-Linearity error, the effect of offset, gain and quantization errors is not included.

Note: Bisector characteristic is obtained drawing a line from 1/2 LSB before the first step of the real characteristic, and 1/2 LSB after the last step again of the real characteristic.

19.5.1 Total unadjusted error

The Total Unadjusted Error specifies the maximum deviation from the ideal characteristic: the number provided in the datasheet represents the maximum error with respect to the entire characteristic. It is a combination of the Offset, Gain and Integral Linearity errors. The different errors may compensate each other depending on the relative sign of the Offset and Gain errors. Refer to *Figure 164*, see TUE.

Figure 164. A/D conversion characteristic



19.5.2 Analog reference pins

The accuracy of the A/D converter depends on how accurate is its analog reference: a noise in the reference results in at least that much error in a conversion. A low pass filter on the A/D converter reference source (supplied through pins V_{AREF} and V_{AGND}), is recommended in order to clean the signal, minimizing the noise. A simple capacitive bypassing may be sufficient in most of the cases; in presence of high RF noise energy, inductors or ferrite beads may be necessary.

In this architecture, V_{AREF} and V_{AGND} pins represents also the power supply of the analog circuitry of the A/D converter: there is an effective DC current requirement from the reference voltage by the internal resistor string in the R-C DAC array and by the rest of the analog circuitry.

An external resistance on V_{AREF} could introduce error under certain conditions: for this reasons, series resistance are not advisable, and more in general any series devices in the filter network should be designed to minimize the DC resistance.

19.5.3 Analog input pins

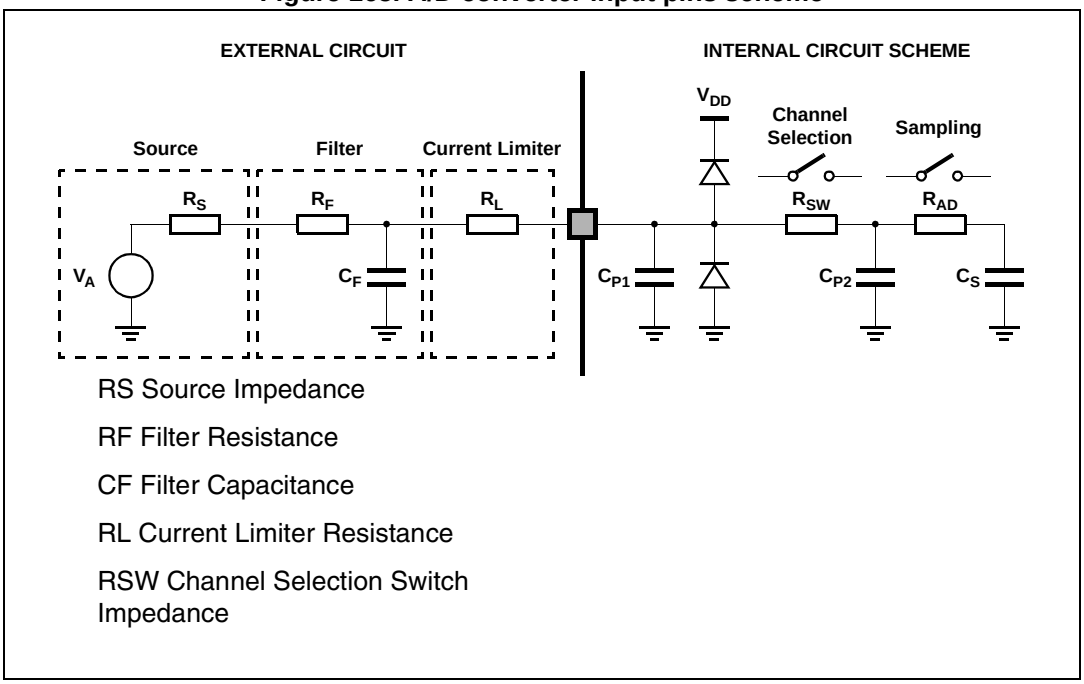
To improve the accuracy of the A/D converter, it is definitively necessary that analog input pins have low AC impedance. Placing a capacitor with good high frequency characteristics at the input pin of the device, can be effective: the capacitor should be as large as possible,

ideally infinite. This capacitor contributes to attenuating the noise present on the input pin; besides, it sources charge during the sampling phase, when the analog signal source is a high-impedance source.

A real filter, can typically be obtained by using a series resistance with a capacitor on the input pin (simple RC Filter). The RC filtering may be limited according to the value of source impedance of the transducer or circuit supplying the analog signal to be measured.

The filter at the input pins must be designed taking into account the dynamic characteristics of the input signal (bandwidth).

Figure 165. A/D converter input pins scheme



Input leakage and external circuit

The series resistor utilized to limit the current to a pin (see R_L in *Figure 165*), in combination with a large source impedance can lead to a degradation of A/D converter accuracy when input leakage is present.

Data about maximum input leakage current at each pin are provided in the datasheet (Electrical Characteristics section). Input leakage is greatest at high operating temperatures, and in general it decreases by one half for each 10°C decrease in temperature.

Considering that, for a 10-bit A/D converter one count is about 5mV (assuming V_{AREF} = 5V), an input leakage of 100nA acting though an R_L = 50kΩ of external resistance leads to an error of exactly one count (5mV); if the resistance were 100kΩ the error would become two counts.

Eventual additional leakage due to external clamping diodes must also be taken into account in computing the total leakage affecting the A/D converter measurements. Another contribution to the total leakage is represented by the charge sharing effects with the sampling capacitance: being C_S substantially a switched capacitance, with a frequency equal to the conversion rate of a single channel (maximum when fixed channel continuous conversion mode is selected), it can be seen as a resistive path to ground. For instance,

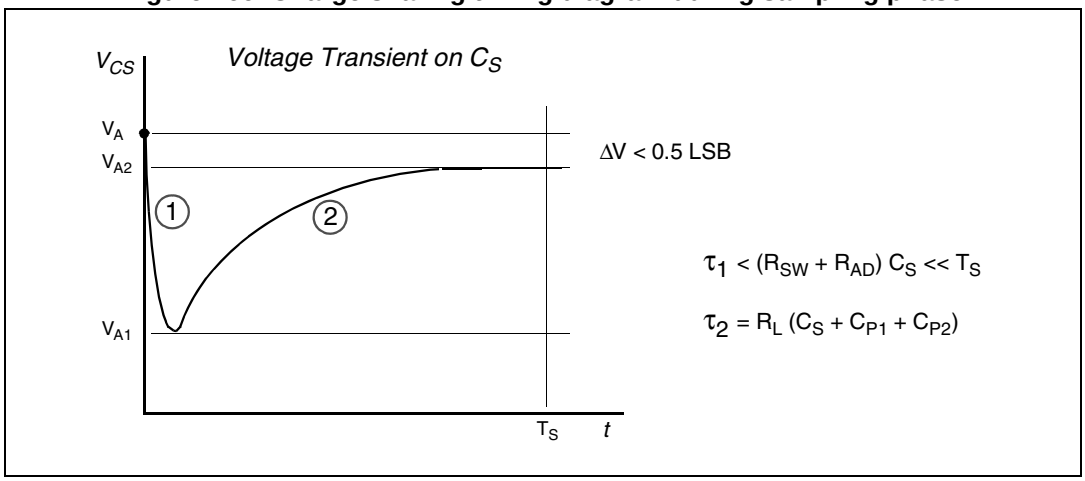
assuming a conversion rate of 250 kHz, with C_S equal to 4pF, a resistance of 1MΩ is obtained ($R_{EQ} = 1 / f_C C_S$, where f_C represents the conversion rate at the considered channel). To minimize the error induced by the voltage partitioning between this resistance (sampled voltage on C_S) and the sum of $R_S + R_F + R_L + R_{SW} + R_{AD}$, the external circuit must be designed to respect the following relation:

$$V_A \cdot \frac{R_S + R_F + R_L + R_{SW} + R_{AD}}{R_{EQ}} < \frac{1}{2}LSB$$

The formula above provides a constraints for external network design, in particular on resistive path.

A second aspect involving the capacitance network should be considered. Assuming the three capacitances C_F, C_{P1} and C_{P2} initially charged at the source voltage V_A (refer to the equivalent circuit reported in *Figure 165*), when the sampling phase is started (A/D switch close), a charge sharing phenomena is installed.

Figure 166. Charge sharing timing diagram during sampling phase



In particular two different transient periods can be distinguished (see *Figure 166*):

- A first and quick charge transfer from the internal capacitance C_{P1} and C_{P2} to the sampling capacitance C_S occurs (C_S is supposed initially completely discharged): considering a worst case (since the time constant in reality would be faster) in which C_{P2} is reported in parallel to C_{P1} (call $C_P = C_{P1} + C_{P2}$), the two capacitance C_P and C_S are in series, and the time constant is:

$$\tau_1 = (R_{SW} + R_{AD}) \cdot \frac{C_P \cdot C_S}{C_P + C_S}$$

- This relation can again be simplified considering only C_S as an additional worst condition. In reality, the transient is faster, but the A/D Converter circuitry has been

designed to be robust also in the very worst case: the sampling time T_S is always much longer than the internal time constant:

$$\tau_1 < (R_{SW} + R_{AD}) \cdot C_S << T_S$$

- The charge of C_{P1} and C_{P2} is redistributed also on C_S, determining a new value of the voltage V_{A1} on the capacitance according to the following equation:

$$V_{A1} \cdot (C_S + C_{P1} + C_{P2}) = V_A \cdot (C_{P1} + C_{P2})$$

- A second charge transfer involves also C_F (that is typically bigger than the on-chip capacitances) through the resistance R_L: again considering the worst case in which C_{P2} and C_S were in parallel to C_{P1} (since the time constant in reality would be faster), the time constant is:

$$\tau_2 < R_L \cdot (C_S + C_{P1} + C_{P2})$$

- In this case, the time constant depends on the external circuit: in particular imposing that the transient is completed well before the end of sampling time T_S, a constraints on R_L sizing is obtained:

$$10 \cdot \tau_2 = 10 \cdot R_L \cdot (C_S + C_{P1} + C_{P2}) \leq T_S$$

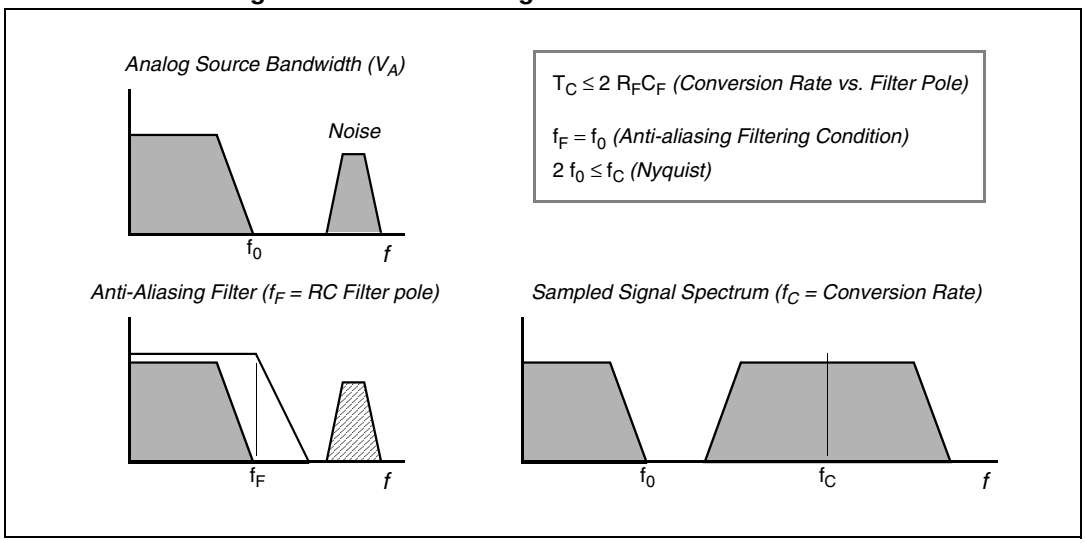
- Of course, R_L should be sized also according to the current limitation constraints, in combination with R_S (source impedance) and R_F (filter resistance). Being C_F definitively bigger than C_{P1}, C_{P2} and C_S, then the final voltage V_{A2} (at the end of the charge transfer transient) will be much higher than V_{A1}. The following equation must be respected (charge balance assuming now C_S already charged at V_{A1}):

$$V_{A2} \cdot (C_S + C_{P1} + C_{P2} + C_F) = V_A \cdot C_F + V_{A1} \cdot (C_{P1} + C_{P2} + C_S)$$

The two transients above are not influenced by the voltage source that, due to the presence of the R_FC_F filter, is not able to provide the extra charge to compensate the voltage drop on C_S with respect to the ideal source V_A; the time constant R_FC_F of the filter is very high with respect to the sampling time (T_S). The filter is typically designed to act as anti-aliasing (see *Figure 167*).

Calling f_0 the bandwidth of the source signal (and as a consequence the cut-off frequency of the anti-aliasing filter, f_F), according to Nyquist theorem the conversion rate f_C must be at least $2f_0$; it means that the constant time of the filter is greater than or at least equal to twice the conversion period (T_C). Again the conversion period T_C is longer than the sampling time T_S, which is just a portion of it, even when fixed channel continuous conversion mode is selected (fastest conversion rate at a specific channel): in conclusion it is evident that the time constant of the filter R_FC_F is definitively much higher than the sampling time T_S, so the charge level on C_S cannot be modified by the analog signal source during the time in which the sampling switch is closed.

Figure 167. Anti-aliasing filter and conversion rate



The considerations above lead to impose new constraints to the external circuit, to reduce the accuracy error due to the voltage drop on C_S; from the two charge balance equations above, it is simple to derive the following relation between the ideal and real sampled voltage on C_S:

$$\frac{V_A}{V_{A2}} = \frac{C_{P1} + C_{P2} + C_F}{C_{P1} + C_{P2} + C_F + C_S}$$

From this formula, in the worst case (when V_A is maximum, that is for instance 5V), assuming to accept a maximum error of half a count (~2.44mV), it is immediately evident a constraints on C_F value:

$$C_F > 2048 \cdot C_S$$

In the next section an example of how to design the external network is provided, assuming some reasonable values for the internal parameters and making hypothesis on the characteristics of the analog signal to be sampled.

19.5.4 Example of external network sizing

The following hypothesis are formulated in order to proceed in designing the external network on A/D Converter input pins:

- Analog Signal Source Bandwidth (f_0): 10 kHz
- Conversion Rate (f_C): 25 kHz
- Sampling Time (T_S): 1µs
- Pin Input Capacitance (C_{P1}): 5pF
- Pin Input Routing Capacitance (C_{P2}): 1pF
- Sampling Capacitance (C_S): 4pF
- Maximum Input Current Injection (I_{INJ}): 3mA
- Maximum Analog Source Voltage ($V_{AM)}$: 12V
- Analog Source Impedance (R_S): 100Ω
- Channel Switch Resistance (R_{SW}): 500Ω
- Sampling Switch Resistance (R_{AD}): 200Ω

1. Supposing to design the filter with the pole exactly at the maximum frequency of the signal, the time constant of the filter is:

$$R_C C_F = \frac{1}{2\pi f_0} = 15.9\mu s$$

2. Using the relation between C_F and C_S and taking some margin (4000 instead of 2048), it is possible to define C_F:

$$C_F = 4000 \cdot C_S = 16nF$$

3. As a consequence of step 1 and 2, RC can be chosen:

$$R_F = \frac{1}{2\pi f_0 C_F} = 995\Omega \cong 1k\Omega$$

4. Considering the current injection limitation and supposing that the source can go up to 12V, the total series resistance can be defined as:

$$R_S + R_F + R_L = \frac{V_{AM}}{I_{INJ}} = 4k\Omega$$

5. from which is now simple to define the value of R_L:

$$R_L = \frac{V_{AM}}{I_{INJ}} - R_F - R_S = 2.9k\Omega$$

6. Now the three element of the external circuit R_F, C_F and R_L are defined. Some conditions discussed in the previous paragraphs have been used to size the component, the other must now be verified. The relation which allow to minimize the

accuracy error introduced by the switched capacitance equivalent resistance is in this case:

$$R_{EQ} = \frac{1}{f_C C_S} = 10M\Omega$$

7. So the error due to the voltage partitioning between the real resistive path and C_S is less then half a count (considering the worst case when V_A = 5V):

$$V_A \cdot \frac{R_S + R_F + R_L + R_{SW} + R_{AD}}{R_{EQ}} = 2.35mV < \frac{1}{2}LSB$$

8. The other conditions to be verified is the time constants of the transients are really and significantly shorter than the sampling period duration T_S:

$$\tau_1 = (R_{SW} + R_{AD}) \cdot C_S = 2.8ns << T_S = 1\mu s$$

$$10 \cdot \tau_2 = 10 \cdot R_L \cdot (C_S + C_{P1} + C_{P2}) = 290ns < T_S = 1\mu s$$

For complete set of parameters characterizing the ST10F276 A/D converter equivalent circuit, refer to the datasheet.

20 I²C interface

The I²C is enabled by setting XPEN, bit 2 of the SYSCON register and bit XI2CEN of XPERCON register. Once this is done, pins P4.4 and P4.7 becomes fully dedicated to I²C interface and all the other alternate functions are bypassed (external memory and CAN2 functions). The pins are also automatically configured as Open-Drain as requested by the I²C bus standard. The Port4 control registers P4, DP4 and ODP4 can no longer control P4.7 and P4.4 pin configuration: writing in the bits corresponding to P4.4 and P4.7 in these registers has no effect on pins activity.

The I²C Bus Interface serves as an interface between the microcontroller and a serial I²C bus. It provides both multi-master and slave functions, and controls all I²C bus-specific sequencing, protocol, arbitration and timing. It supports fast I²C mode (400 kHz).

20.1 Main features

- Parallel-bus/I²C protocol converter
- Multi-master capability
- 7-bit/10-bit addressing
- Transmitter/receiver flag
- End-of-byte transmission flag
- Transfer problem detection
- Standard/fast I²C mode

I²C timing diagram I²C master features:

- Clock generation
- I²C bus busy flag
- Arbitration lost flag
- End of byte transmission flag
- Transmitter/receiver flag
- Start bit detection flag
- Start and stop generation

I²C slave features:

- Stop bit detection
- I²C bus busy flag
- Detection of misplaced start or stop condition
- Programmable I²C address detection
- Transfer problem detection
- End-of-byte transmission flag
- Transmitter/receiver flag

20.2 General description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa, using either an interrupt or polled handshake. The interrupts are enabled or disabled by software. The interface is connected to the I²C bus by a data pin (SDA) and by a clock pin (SCL). It can be connected both with a standard I²C bus and a Fast I²C bus. This selection is made by software.

20.2.1 Mode selection

The interface can operate in the four following modes:

- Slave transmitter/receiver
- Master transmitter/receiver

By default, it operates in slave mode.

The interface automatically switches from slave to master after it generates a START condition and from master to slave in case of arbitration loss or a STOP generation, allowing then Multi-Master capability.

20.2.2 Communication flow

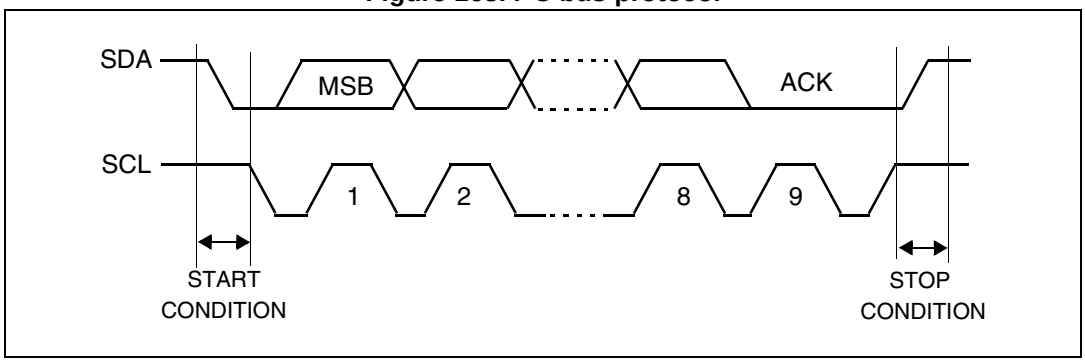
In Master mode, it initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by hardware as soon as the Master mode is selected.

In Slave mode, the interface is capable of recognizing its own address (7- or 10-bit), and the General Call address. The General Call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the start condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode.

A 9th clock pulse follows the eight clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to *Figure 168*.

Figure 168. I²C bus protocol



Acknowledge may be enabled and disabled by software.

The I²C interface address and/or general call address can be selected by software.

The speed of the I²C interface may be selected between Standard (0-100kHz) and Fast I²C (100-400kHz).

20.2.3 SDA/SCL line control

Transmitter mode: the interface holds the clock line low before transmission to wait for the microcontroller to write the byte in the Data Register.

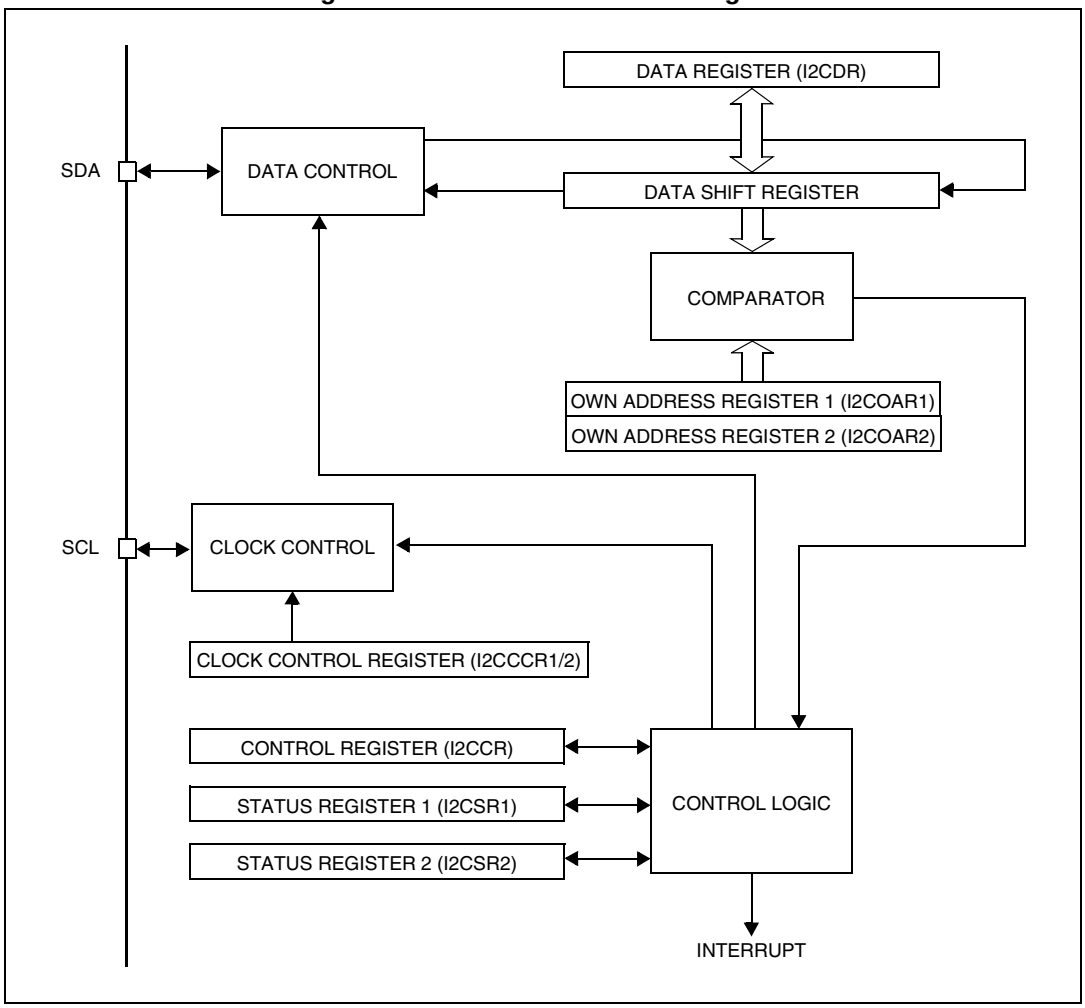
Receiver mode: the interface holds the clock line low after reception to wait for the microcontroller to read the byte in the Data Register.

The SCL frequency (F_{SCL}) is controlled by a programmable clock divider which depends on the I²C bus mode.

When the I²C cell is enabled by setting bit XI2CEN in XPERCON register, pins P4.4 and P4.7 (where SCL and SDA are respectively mapped as alternate functions) are automatically configured as bidirectional open-drain: the value of the external pull-up resistor depends on the application. P4, DP4 and ODP4 cannot influence the pin configuration.

When the I²C cell is disabled (clearing bit XI2CEN), P4.4 and P4.7 pins are standard I/O controlled by P4, DP4 and ODP4.

Figure 169. I²C interface block diagram



20.3 Functional description

Refer to the I2CCR, I2CSR1 and I2CSR2 registers in *Section 20.5: Register description* for the bit definitions.

By default the I²C interface operates in Slave mode (M/SL bit is cleared) except when it initiates a transmit or receive sequence.

First the interface frequency must be configured using the FRx bits in the I2COAR2 register.

20.3.1 Slave mode

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register; then it is compared with the address of the interface or the General Call address (if selected by software).

Note: In 10-bit addressing mode, the comparison includes the header sequence (11110xx0) and the two most significant bits of the address.

Header matched (10-bit mode only): the interface generates an acknowledge pulse if the ACK bit is set.

Address not matched: the interface ignores it and waits for another Start condition.

Address matched: the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set.
- EVF and ADSL bits are set with an interrupt if the ITE bit is set.

Then the interface waits for a read of the I2CSR1 register, **holding the SCL line low** (see *Figure 170* - Transfer sequencing EV1).

In 10-bit mode, after receiving the address sequence the slave is always in receive mode. It will enter transmit mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

Slave receiver

Following the address reception and after I2CSR1 register has been read, the slave receives bytes from the SDA line into the I2CDR register via the internal shift register. After each byte the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set
- EVF and BTF bits are set with an interrupt if the ITE bit is set.

Then the interface waits for a read of the I2CDR register, **holding the SCL line low** (see *Figure 170* - Transfer sequencing EV2).

Slave transmitter

Following the address reception and after I2CSR1 register has been read, the slave sends bytes from the I2CDR register to the SDA line via the internal shift register.

The slave waits for a write to the I2CDR register, **holding the SCL line low** (see *Figure 170* - Transfer sequencing EV3).

When the acknowledge pulse is received:

- The EVF and BTF bits are set by hardware with an interrupt if the ITE bit is set.

Closing slave communication

After the last data byte is transferred a Stop Condition is generated by the master. The interface detects this condition and sets:

- EVF and STOPF bits with an interrupt if the ITE bit is set.

Then the interface waits for a read of the I2CSR2 register (see *Figure 170* - Transfer sequencing EV4).

Error cases

- **BERR**: Detection of a Stop or a Start condition during a byte transfer. In this case, the EVF and the BERR bits are set with an interrupt if the ITE bit is set.
  If it is a Stop then the interface discards the data, releases the lines and waits for another Start condition.

If it is a Start then the interface discards the data and waits for the next slave address on the bus.

- **AF**: Detection of a non-acknowledge bit. In this case, the EVF and AF bits are set with an interrupt if the ITE bit is set.

Note: In both cases, SCL line is not held low; however, SDA line can remain low due to possible «0» bits transmitted last. It is then necessary to release both lines by software.

How to release the SDA / SCL lines

Set and subsequently clear the STOP bit while BTF is set. The SDA/SCL lines are released after the transfer of the current byte.

20.3.2 Master mode

To switch from default Slave mode to Master mode a Start condition generation is needed.

Start condition

Setting the START bit while the BUSY bit is cleared causes the interface to switch to Master mode (M/SL bit set) and generates a Start condition.

Once the Start condition is sent:

– The EVF and SB bits are set by hardware with an interrupt if the ITE bit is set.

Then the master waits for a write to the I2CDR register with the Slave address, **holding the SCL line low** (see *Figure 170* - Transfer sequencing EV5).

Slave address transmission

Then the slave address is sent to the SDA line via the internal shift register.

In 7-bit addressing mode, one address byte is sent.

In 10-bit addressing mode, sending the first byte including the header sequence causes the following event:

- The EVF and ADD10 bits are set by hardware with interrupt generation if the ITE bit is set.

Then the master waits for a read of the I2CSR1 register followed by a write in the I2CDR register, **holding the SCL line low** (see *Figure 170* - Transfer sequencing EV9).

Then the second address byte is sent by the interface.

After completion of this transfer (and acknowledge from the slave if the ACK bit is set):

- The EVF and ENDAD bits are set by hardware with interrupt generation if the ITE bit is set.

Then the master waits for a read of the I2CSR1 register followed by a write in the I2CCR register (for example set PE bit), **holding the SCL line low** (see *Figure 170* - Transfer sequencing EV6).

Next the master must enter Receiver or Transmitter mode.

Note: In 10-bit addressing mode, to switch the master to Receiver mode, software must generate a repeated Start condition and re-send the header sequence with the least significant bit set (11110xx1).

Master receiver

Following the address transmission and after I2CSR1 and I2CCR registers have been accessed, the master receives bytes from the SDA line into the I2CDR register via the internal shift register. After each byte the interface generates in sequence:

- Acknowledge pulse if the ACK bit is set
- EVF and BTF bits are set by hardware with an interrupt if the ITE bit is set.

Then the interface waits for a read of the I2CDR register, **holding the SCL line low** (see *Figure 170* Transfer sequencing EV7).

To close the communication: before reading the last byte from the I2CDR register, set the STOP bit to generate the Stop condition. The interface goes automatically back to slave mode (M/SL bit cleared).

Note: In order to generate the non-acknowledge pulse after the last received data byte, the ACK bit must be cleared just before reading the second last data byte.

Master transmitter

Following the address transmission and after I2CSR1 register has been read, the master sends bytes from the I2CDR register to the SDA line via the internal shift register.

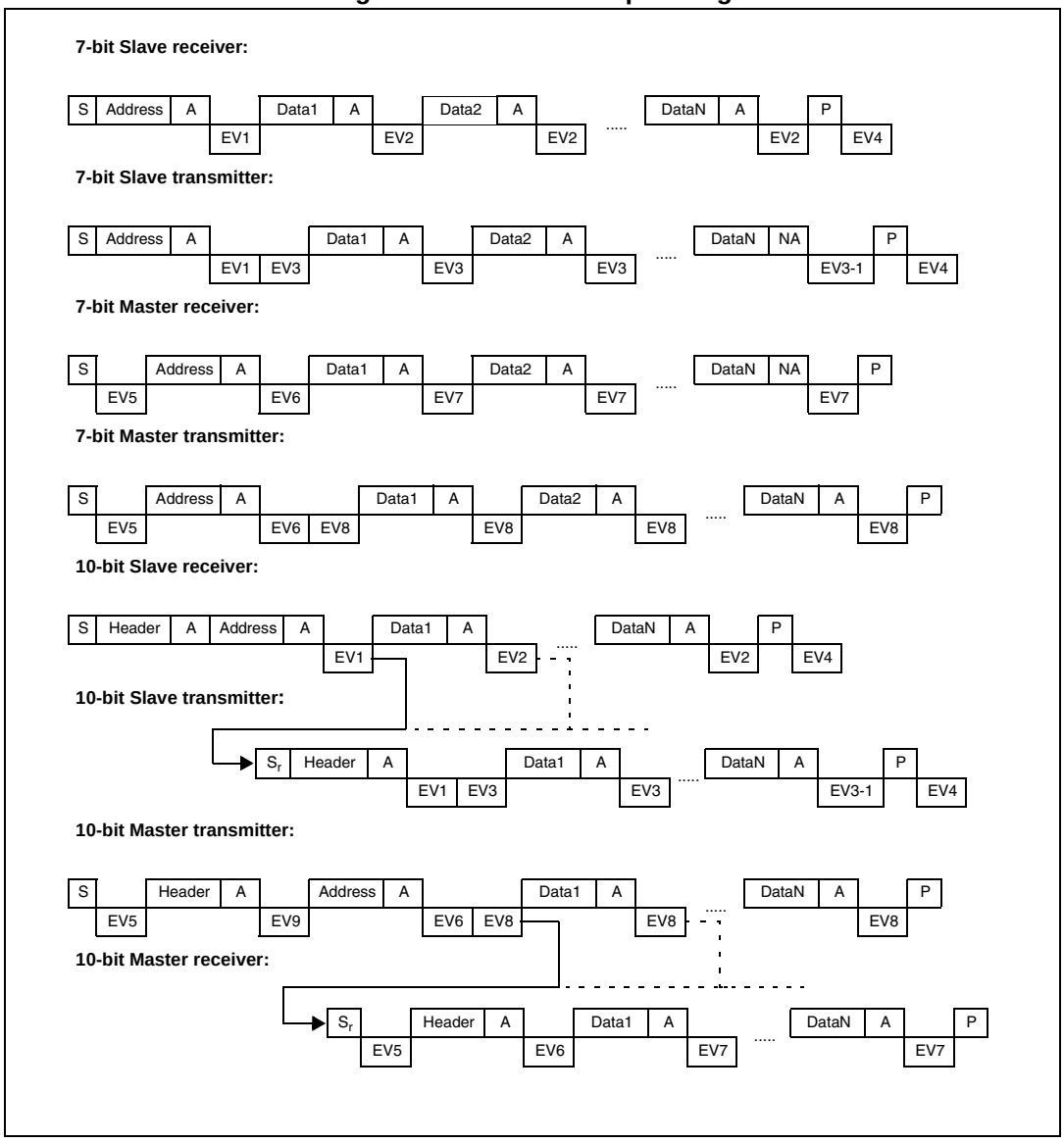
The master waits for a write in the I2CDR register, **holding the SCL line low** (see *Figure 170* - Transfer sequencing EV8).

When the acknowledge bit is received, the interface sets:

- EVF and BTF bits with an interrupt if the ITE bit is set.

To close the communication: after writing the last byte to the I2CDR register, set the STOP bit to generate the Stop condition. The interface goes automatically back to slave mode (M/SL bit cleared).

Figure 170. Transfer sequencing



Legend:

S	Start
S_r	Repeated Start
P	Stop
A	Acknowledge
NA	Non-acknowledge
EVx	Event (with interrupt if ITE = 1)
EV1	EVF = 1, ADSL = 1, cleared by reading I2CSR1 register.
EV2	EVF = 1, BTF = 1, cleared by reading I2CDR register.
EV3	EVF = 1, BTF = 1, cleared by reading I2CSR1 register followed by writing I2CDR register.

EV3-1 EVF = 1, AF = 1, BTF = 1; AF is cleared by reading I2CSR2 register. BTF is cleared by releasing the lines (STOP = 1, STOP = 0) or by writing I2CDR register (I2CDR = FFh).

Note: If lines are released by STOP = 1, STOP = 0, the subsequent EV4 is not seen.

EV4 EVF = 1, STOPF = 1, cleared by reading I2CSR2 register.

EV5 EVF = 1, SB = 1, cleared by reading I2CSR1 register followed by writing I2CDR register.

EV6 EVF = 1, ENDAD = 1, cleared by reading I2CSR2 register followed by writing I2CCR register (for example PE = 1).

EV7 EVF = 1, BTF = 1, cleared by reading I2CDR register.

EV8 EVF = 1, BTF = 1, cleared by writing I2CDR register.

EV9 EVF = 1, ADD10 = 1, cleared by reading I2CSR1 register followed by writing I2CDR register.

Error cases

- **BERR**: Detection of a Stop or a Start condition during a byte transfer. In this case, the EVF and BERR bits are set by hardware with an interrupt if ITE is set.
- **AF**: Detection of a non-acknowledge bit. In this case, the EVF and AF bits are set by hardware with an interrupt if the ITE bit is set. To resume, set the START or STOP bit.
- **ARLO:** Detection of an arbitration lost condition or misplaced data transition during high phase of SCL.
  In these cases the ARLO bit is set by hardware (with an interrupt if the ITE bit is set) and the interface goes automatically back to slave mode (the M/SL bit is cleared).

Note: In all these cases, the SCL line is not held low; however, the SDA line can remain low due to possible «0» bits transmitted last. It is then necessary to release both lines by software.

20.4 Interrupts

There are three different types of interrupt that the module can generate:

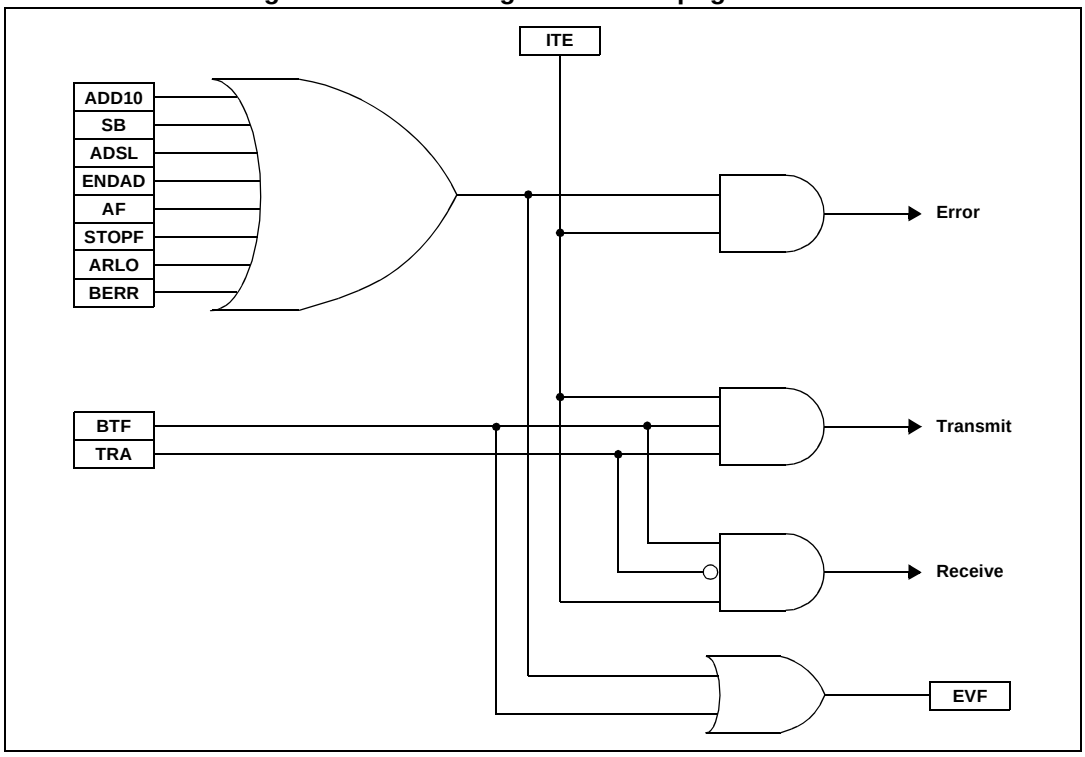
- requests related to bus events, like start or stop events, arbitration lost, etc.;
- requests related to data transmission;
- requests related to data reception;

These requests are issued to the interrupt controller by three different lines, and identified as Error, Transmit and Receive interrupt lines.

Table 57. Interrupt event summary

Line	Interrupt event	Event flag
Error	10-bit Address Sent Event (Master mode)	ADD10
	Start Bit Generation Event (Master mode)	SB
	Address Matched Event (Slave mode)	ADSL
	Address Byte Transmission Event (Master Mode)	ENDAD
	Acknowledge Failure Event	AF
	Stop Detection Event (Slave mode)	STOPF
	Arbitration Lost Event (Multi-master configuration)	ARLO
	Bus Error Event	BERR
Transmit	Byte Transmission Event	BTF with TRA = 1
Receive	Byte Reception Event	BTF with TRA = 0

Figure 171. Event flags and interrupt generation



Up to four interrupt control registers (XIRxSEL, x = 0, 1, 2, 3) are provided in order to select the source of the XBUS interrupt: the transmit interrupt, the receive interrupt and the error interrupt of I²C Interface are linked to the one of the XPxIC registers (x = 0, 1, 2, 3). In particular, the three interrupt lines are available on the following interrupt vectors:

- Receive XP0INT XP1INT XP2INT
- Transmit XP0INT XP1INT XP2INT
- Error XP3INT

Refer to *Section 5.7: X-peripheral interrupt on page 117* for details.

When interruptible Power Down mode is entered, I²C SCL line (P4.4) can be used to wake-up the device from low power mode without resetting it, restarting the application from where it was stopped at the execution of PWRDN instruction.

Again, refer to *Section 5.6.1: Fast external interrupts on page 115* for further details.

20.5 Register description

I2CCR (EA00h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–										PE	ENGC	START	ACK	STOP	ITE
										RW	RW	RW	RW	RW	RW

Bit	Function
ITE	**Interrupt Enable** This bit is set and cleared by software and cleared by hardware when the interface is disabled (PE = 0). ‘0’: Interrupts disabled. ‘1’: Interrupts enabled. Refer to *Figure 171* for the relationship between the events and the interrupt. SCL is held low when the ADD10, SB, BTF or ADSL flags or an EV6 event (see *Figure 170*) is detected.
STOP	**Generation of a Stop condition** This bit is set and cleared by software. It is also cleared by hardware in master mode. This bit is not cleared when the interface is disabled (PE = 0). – In master mode: ‘0’: No stop generation. ‘1’: Stop generation after the current byte transfer or after the current Start condition is sent. The STOP bit is cleared by hardware when the Stop condition is sent. – In slave mode: ‘0’: No stop generation. ‘1’: Release the SCL and SDA lines after the current byte transfer (BTF = 1). In this mode the STOP bit has to be cleared by software.
ACK	**Acknowledge Enable** This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE = 0). ‘0’: No acknowledge returned ‘1’: Acknowledge returned after an address byte or a data byte is received
START	**Generation of a Start condition** This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE = 0) or when the Start condition is sent (with interrupt generation if ITE = 1). – In master mode: ‘0’: No start generation. ‘1’: Repeated start generation. – In slave mode: ‘0’: No start generation. ‘1’: Start generation when the bus is free.

Bit	Function
ENGC	**Enable General Call** This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE = 0). The 00h General Call address is acknowledged (01h ignored). ‘0’: General Call disabled. ‘1’: General Call enabled.
PE	**Peripheral Enable** This bit is set and cleared by software. ‘0’: Peripheral disabled ‘1’: Master/Slave capability *Notes:* – ‘0’: all the bits of the I2CCR, I2CSR1 and I2CSR2 registers except the STOP bit are reset. – ‘1’: the corresponding I/O pins are selected by hardware as alternate functions. To enable the I²C interface, write the I2CCR register **TWICE** with PE = 1 as the first write only activates the interface (only PE is set).

I2CSR1 (EA02h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–								EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB
								R	R	R	R	R	R	R	R

Bit	Function
SB	**Start bit (Master mode)** This bit is set by hardware as soon as the Start condition is generated (following a write START = 1). An interrupt is generated if ITE = 1. It is cleared by software reading I2CSR1 register followed by writing the address byte in I2CDR register. It is also cleared by hardware when the interface is disabled (PE = 0). ‘0’: No Start condition. ‘1’: Start condition generated.
M/SL	**Master/Slave** This bit is set by hardware as soon as the interface is in Master mode (writing START = 1). It is cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO = 1). It is also cleared when the interface is disabled (PE = 0). ‘0’: Slave mode ‘1’: Master mode
ADSL	**Address matched (Slave mode)** This bit is set by hardware as soon as the received slave address matched with the OAR register content or a general call is recognized. An interrupt is generated if ITE = 1. It is cleared by software reading I2CSR1 register or by hardware when the interface is disabled (PE = 0). The SCL line is held low while ADSL = 1. ‘0’: Address mismatched or not received. ‘1’: Received address matched.

Bit	Function
BTF	**Byte transfer finished** This bit is set by hardware as soon as a byte is correctly received or transmitted with interrupt generation if ITE = 1. It is cleared by software reading I2CSR1 register followed by a read or write of I2CDR register. It is also cleared by hardware when the interface is disabled (PE = 0). – Following a byte transmission, this bit is set after reception of the acknowledge clock pulse. In case an address byte is sent, this bit is set only after the EV6 event (see *Figure 170*). BTF is cleared by writing the next byte in I2CDR register. – Following a byte reception, this bit is set after transmission of the acknowledge clock pulse if ACK = 1. BTF is cleared the byte from I2CDR register. The SCL line is held low while BTF = 1. ‘0’: Byte transfer not done ‘1’: Byte transfer succeeded
BUSY	**Bus busy** This bit is set by hardware on detection of a Start condition and cleared by hardware on detection of a Stop condition. It indicates a communication in progress on the bus. This information is still updated when the interface is disabled (PE = 0). ‘0’: No communication on the bus ‘1’: Communication ongoing on the bus
TRA	**Transmitter/Receiver** When BTF is set, TRA = 1 if a data byte has been transmitted. It is cleared automatically when BTF is cleared. It is also cleared by hardware after detection of Stop condition (STOPF = 1), loss of bus arbitration (ARLO = 1) or when the interface is disabled (PE = 0). ‘0’: Data byte received (if BTF = 1). ‘1’: Data byte transmitted.
ADD10	**10-bit addressing in Master mode** This bit is set by hardware when the master has sent the first byte in 10-bit address mode. It is cleared by software reading I2CSR1 register followed by a write in the I2CDR register of the second address byte. It is also cleared by hardware when the peripheral is disabled (PE = 0). ‘0’: No ADD10 event occurred. ‘1’: Master has sent first address byte (header).
EVF	**Event Flag** This bit is set by hardware as soon as an event occurs. It is cleared by software reading I2CSR2 register in case of error event or as described in *Figure 170*. It is also cleared by hardware when the interface is disabled (PE = 0). ‘0’: No event ‘1’: One of the following events has occurred: – BTF = 1 (Byte received or transmitted) – ADSL = 1 (Address matched in Slave mode while ACK = 1) – SB = 1 (Start condition generated in Master mode) – AF = 1 (No acknowledge received after byte transmission) – STOPF = 1 (Stop condition detected in Slave mode) – ARLO = 1 (Arbitration lost in Master mode or misplaced data transition when SCL is high) – BERR = 1 (Bus error, misplaced Start or Stop condition detected) – ADD10 = 1 (Master has sent header byte) – ENDAD = 1 (Address byte successfully transmitted in Master mode)

I2CSR2 (EA04h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–										ENDAD	AF	STOPF	ARLO	BERR	GCAL
										R	R	R	R	R	R

Bit	Function
GCAL	**General Call (Slave mode)** This bit is set by hardware when a general call address is detected on the bus while ENGC = 1. It is cleared by hardware detecting a Stop condition (STOPF = 1) or when the interface is disabled (PE = 0). ‘0’: No general call address detected on bus ‘1’: general call address detected on bus
BERR	**Bus Error** This bit is set by hardware when the interface detects a misplaced Start or Stop condition at falling edge of SCL, or on the second data transition occurring during the high phase of SCL. An interrupt is generated if ITE = 1. It is cleared by software reading I2CSR2 register or by hardware when the interface is disabled (PE = 0). The SCL line is not held low while BERR = 1. ‘0’: No misplaced Start or Stop condition ‘1’: Misplaced Start or Stop condition
ARLO	**Arbitration Lost** This bit is set by hardware when the interface loses the arbitration of the bus to another master. Besides, it is set when a misplaced data transition occurs during the high phase of SCL. An interrupt is generated if ITE = 1. It is cleared by software reading I2CSR2 register or by hardware when the interface is disabled (PE = 0). After an ARLO event the interface switches back automatically to Slave mode (M/SL = 0). The SCL line is not held low while ARLO = 1. ‘0’: No arbitration lost detected ‘1’: Arbitration lost detected
STOPF	**Stop Detection (Slave Mode)** This bit is set by hardware when a Stop condition is detected on the bus after an acknowledge (if ACK = 1). An interrupt is generated if ITE = 1. It is cleared by software reading I2CSR2 register or by hardware when the interface is disabled (PE = 0). The SCL line is not held low while STOPF = 1. ‘0’: No Stop condition detected ‘1’: Stop condition detected

Bit	Function
AF	**Acknowledge Failure** This bit is set by hardware when no acknowledge is returned. An interrupt is generated if ITE = 1. It is cleared by software by reading I2CSR2 register or by hardware when the interface is disabled (PE = 0). The SCL line is not held low while AF = 1. ‘0’: No acknowledge failure ‘1’: Acknowledge failure
ENDAD	**End of address transmission** This bit is set by hardware when - *7-bit addressing mode*: the address byte has been transmitted; - *10-bit addressing mode*: the MSB and the LSB have been transmitted during the addressing phase. When the master needs to receive data from the slave, it has to send just the MSB of the slave once again; hence the ENDAD flag is set, without waiting for the LSB of the address. It is cleared by software by reading I2CSR2 and a following write to the I2CCR or by hardware when the interface is disabled (PE = 0). ‘0’: No end of address transmission ‘1’: End of address transmission

I2CCCR1 (EA06h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–								FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0
								RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
CC(6:0)	**12-bit clock divider** CC(6:0) bits along with CC(11:7) of the register I2CCCR2 select the speed of the bus (f_{SCL}) depending on the I²C mode. They are not cleared when the interface is disabled (PE = 0). – Standard mode (FM/SM = 0): $f_{SCL} \leq 100$ kHz $f_{SCL} = f_{CPU} / (2 \times CC(11:0) + 7)$ – Fast mode (FM/SM = 1): $f_{SCL} > 100$ kHz $f_{SCL} = f_{CPU} / (3 \times CC(11:0) + 9)$ Note: The programmed f_{SCL} assumes no load on SCL and SDA lines.
FM/SL	**Fast/Standard I²C Mode** This bit is set and cleared by software. It is not cleared when the interface is disabled (PE = 0). ‘0’: Standard I²C mode ‘1’: Fast I²C mode

I2COAR1 (EA08h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–								ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
								RW	RW	RW	RW	RW	RW	RW	RW

7-bit addressing mode

Bit	Function
ADD0	**Address direction bit** This bit is don't care, the interface acknowledges either 0 or 1. It is not cleared when the interface is disabled (PE = 0). *Note: Address 01h is always ignored.*
ADD(7:1)	**Interface address** These bits define the I²C bus address of the interface. They are not cleared when the interface is disabled (PE = 0).

10-bit addressing mode

Bit	Function
ADD(7:0)	**Interface address** These are the least significant bits of the I²C bus address of the interface. They are not cleared when the interface is disabled (PE = 0).

I2COAR2 (EA0Ah) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–								FR2	FR1	FR0	–		ADD9	ADD8	–
								RW	RW	RW			RW	RW	

Bit	Function
ADD(9:8)	**Interface address** These are the most significant bits of the I²C bus address of the interface (10-bit mode only). They are not cleared when the interface is disabled (PE = 0).
FR(2:0)	**Frequency Bits** These bits are set by software only when the interface is disabled (PE = 0). To configure the interface to I²C specified delays select the value corresponding to the system frequency f_{CPU}. f_{CPU} range (MHz) FR2 FR1 FR0 3.3 - 10.0 0 0 0 10.0 - 16.7 0 0 1 16.7 - 26.7 0 1 0 26.7 - 40.0 0 1 1 40.0 - 53.3 1 0 0 53.3 - 66.0 1 0 1 66.0 - 80.0 1 1 0 80.0 - 100.0 1 1 1

I2CDR (EA0Ch) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–								D7	D6	D5	D4	D3	D2	D1	D0
								RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
D(7:0)	**8-bit Data Register** These bits contain the received byte or the byte to be transmitted on the bus. – Transmitter mode: Byte transmission starts automatically when the software writes in the I2CDR register. – Receiver mode: The first data byte is received automatically in the I2CDR register using the least significant bit of the address. Then, the following data bytes are received one by one after reading the I2CDR register.

I2CCCR2 (EA0Eh) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–											CC11	CC10	CC9	CC8	CC7
											RW	RW	RW	RW	RW

Bit	Function
CC(11:7)	**Additional 5-bit clock divider** These bits along with those of the Clock Control Register I2CCCR1 select the speed of the bus (f_{SCL}) depending on the I²C mode. They are not cleared when the interface is disabled (PE = 0)

21 CAN modules

The two integrated CAN modules (CAN1 and CAN2) are identical and handle the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part A and B (active).

The module is based on C-CAN module characteristics. The following system resources are used to interface the module with the ST10 core:

- Interrupt of CAN1 and CAN2 are connected to the XBUS interrupt lines: refer to next *Section 21.2: Interrupt* for details.
- Both CAN modules have to be selected, before the bit XPEN is set in SYSCON register, by setting the proper bit in XPERCON register.
- After reset, CAN1 is enabled by default (see Reset value of XPERCON register). The CAN2 on the contrary is not enabled.

21.1 Memory and pin mapping

21.1.1 CAN1 mapping

Address range 00'EF00h - 00'EFFFh is reserved for the CAN1 Module access. The CAN1 is enabled by setting bit XPEN of the SYSCON register and bit 0 of XPERCON register. Accesses to the CAN Module use demultiplexed addresses and a 16-bit data bus (only word accesses are possible). Two waitstates give an access time of 62.5ns @64MHz CPU clock. No tristate waitstate is used.

After reset, CAN1 is enabled by default (see Reset value of XPERCON register). It is available on pins P4.5 and P4.6.

21.1.2 CAN2 mapping

Address range 00'EE00h - 00'EEFFh is reserved for the CAN2 Module access. The CAN2 is enabled by setting bit XPEN of the SYSCON register and bit 1 of the XPERCON register. Accesses to the CAN Module use demultiplexed addresses and a 16-bit data bus (only word accesses are possible). Two waitstates give an access time of 62.5ns @64MHz CPU clock. No tristate waitstate is used.

After reset, CAN2 is disabled by default (see Reset value of XPERCON register). Once enabled, it is available on pins P4.4 and P4.7. When I^2C interface is enabled as well (bit XI2CEN in XPERCON register), CAN2 is not available on P4.4 and P4.7, since for I^2C a higher priority has been set by hardware option.

Note: If one or the two CAN modules are used, Port4 cannot be programmed to output all eight segment address lines. Thus, only four segment address lines can be used, reducing the external memory space to 5 Mbytes (1 Mbyte per $\overline{CS}$ line).

21.1.3 Register summary

In the tables below, the CAN modules register mapping is summarized.

Table 58. CAN1 register mapping

Name	Physical address	Description	Reset value
CAN1CR	EF00h	CAN1: CAN Control Register	0001h
CAN1SR	EF02h	CAN1: Status Register	0000h
CAN1EC	EF04h	CAN1: Error Counter	0000h
CAN1BTR	EF06h	CAN1: Bit Timing Register	2301h
CAN1IR	EF08h	CAN1: Interrupt Register	0000h
CAN1TR	EF0Ah	CAN1: Test Register	00x0h
CAN1BRPER	EF0Ch	CAN1: BRP Extension Register	0000h
CAN1IF1CR	EF10h	CAN1: IF1 Command Request	0001h
CAN1IF1CM	EF12h	CAN1: IF1 Command Mask	0000h
CAN1IF1M1	EF14h	CAN1: IF1 Mask 1	FFFFh
CAN1IF1M2	EF16h	CAN1: IF1 Mask 2	FFFFh
CAN1IF1A1	EF18h	CAN1: IF1 Arbitration 1	0000h
CAN1IF1A2	EF1Ah	CAN1: IF1 Arbitration 2	0000h
CAN1IF1MC	EF1Ch	CAN1: IF1 Message Control	0000h
CAN1IF1DA1	EF1Eh	CAN1: IF1 Data A 1	0000h
CAN1IF1DA2	EF20h	CAN1: IF1 Data A 2	0000h
CAN1IF1DB1	EF22h	CAN1: IF1 Data B 1	0000h
CAN1IF1DB2	EF24h	CAN1: IF1 Data B 2	0000h
CAN1IF2CR	EF40h	CAN1: IF2 Command Request	0001h
CAN1IF2CM	EF42h	CAN1: IF2 Command Mask	0000h
CAN1IF2M1	EF44h	CAN1: IF2 Mask 1	FFFFh
CAN1IF2M2	EF46h	CAN1: IF2 Mask 2	FFFFh
CAN1IF2A1	EF48h	CAN1: IF2 Arbitration 1	0000h
CAN1IF2A2	EF4Ah	CAN1: IF2 Arbitration 2	0000h
CAN1IF2MC	EF4Ch	CAN1: IF2 Message Control	0000h
CAN1IF2DA1	EF4Eh	CAN1: IF2 Data A 1	0000h
CAN1IF2DA2	EF50h	CAN1: IF2 Data A 2	0000h
CAN1IF2DB1	EF52h	CAN1: IF2 Data B 1	0000h
CAN1IF2DB2	EF54h	CAN1: IF2 Data B 2	0000h
CAN1TR1	EF80h	CAN1: Transmission Request 1	0000h
CAN1TR2	EF82h	CAN1: Transmission Request 2	0000h
CAN1ND1	EF90h	CAN1: New Data 1	0000h
CAN1ND2	EF92h	CAN1: New Data 2	0000h
CAN1IP1	EFA0h	CAN1: Interrupt Pending 1	0000h

Table 58. CAN1 register mapping (continued)

Name	Physical address	Description	Reset value
CAN1IP2	EFA2h	CAN1: Interrupt Pending 2	0000h
CAN1MV1	EFB0h	CAN1: Message Valid 1	0000h
CAN1MV2	EFB2h	CAN1: Message Valid 2	0000h

Table 59. CAN2 register mapping

Name	Physical address	Description	Reset value
CAN2CR	EE00h	CAN2: CAN Control Register	0001h
CAN2SR	EE02h	CAN2: Status Register	0000h
CAN2EC	EE04h	CAN2: Error Counter	0000h
CAN2BTR	EE06h	CAN2: Bit Timing Register	2301h
CAN2IR	EE08h	CAN2: Interrupt Register	0000h
CAN2TR	EE0Ah	CAN2: Test Register	00x0h
CAN2BRPER	EE0Ch	CAN2: BRP Extension Register	0000h
CAN2IF1CR	EE10h	CAN2: IF1 Command Request	0001h
CAN2IF1CM	EE12h	CAN2: IF1 Command Mask	0000h
CAN2IF1M1	EE14h	CAN2: IF1 Mask 1	FFFFh
CAN2IF1M2	EE16h	CAN2: IF1 Mask 2	FFFFh
CAN2IF1A1	EE18h	CAN2: IF1 Arbitration 1	0000h
CAN2IF1A2	EE1Ah	CAN2: IF1 Arbitration 2	0000h
CAN2IF1MC	EE1Ch	CAN2: IF1 Message Control	0000h
CAN2IF1DA1	EE1Eh	CAN2: IF1 Data A 1	0000h
CAN2IF1DA2	EE20h	CAN2: IF1 Data A 2	0000h
CAN2IF1DB1	EE22h	CAN2: IF1 Data B 1	0000h
CAN2IF1DB2	EE24h	CAN2: IF1 Data B 2	0000h
CAN2IF2CR	EE40h	CAN2: IF2 Command Request	0001h
CAN2IF2CM	EE42h	CAN2: IF2 Command Mask	0000h
CAN2IF2M1	EE44h	CAN2: IF2 Mask 1	FFFFh
CAN2IF2M2	EE46h	CAN2: IF2 Mask 2	FFFFh
CAN2IF2A1	EE48h	CAN2: IF2 Arbitration 1	0000h
CAN2IF2A2	EE4Ah	CAN2: IF2 Arbitration 2	0000h
CAN2IF2MC	EE4Ch	CAN2: IF2 Message Control	0000h
CAN2IF2DA1	EE4Eh	CAN2: IF2 Data A 1	0000h
CAN2IF2DA2	EE50h	CAN2: IF2 Data A 2	0000h

Table 59. CAN2 register mapping

Name	Physical address	Description	Reset value
CAN2IF2DB1	EE52h	CAN2: IF2 Data B 1	0000h
CAN2IF2DB2	EE54h	CAN2: IF2 Data B 2	0000h
CAN2TR1	EE80h	CAN2: Transmission Request 1	0000h
CAN2TR2	EE82h	CAN2: Transmission Request 2	0000h
CAN2ND1	EE90h	CAN2: New Data 1	0000h
CAN2ND2	EE92h	CAN2: New Data 2	0000h
CAN2IP1	EEA0h	CAN2: Interrupt Pending 1	0000h
CAN2IP2	EEA2h	CAN2: Interrupt Pending 2	0000h
CAN2MV1	EEB0h	CAN2: Message Valid 1	0000h
CAN2MV2	EEB2h	CAN2: Message Valid 2	0000h

21.2 Interrupt

Up to four interrupt control registers (XIRxSEL, x = 0, 1, 2, 3) are provided in order to select the source of the XBUS interrupt: one line for each module is provided and differently linked to one of the XPxIC registers (x = 0, 1, 2, 3). In particular, the two interrupt lines are available on the following interrupt vectors:

- CAN1 XP0INT XP3INT
- CAN2 XP1INT XP3INT

Refer to *Section 5.7: X-peripheral interrupt on page 117* for details.

When interruptible Power Down mode is entered, both CAN1 and CAN2 lines can be used to wake-up the device from low power mode without resetting it, restarting the application from where it was stopped before the execution of PWRDN instruction.

Refer to *Section 5.6.1: Fast external interrupts on page 115*.

21.3 Configuration support

It is possible that both CAN controllers are working on the same CAN bus, supporting together up to 64 message objects. In this configuration, both receive signals and both transmit signals are linked together when using the same CAN transceiver. This configuration is especially supported by providing open drain outputs for the CAN1_Txd and CAN2_TxD signals. The open drain function is controlled with the ODP4 register for port P4: in this way it is possible to connect together P4.4 with P4.5 (receive lines) and P4.6 with P4.7 (transmit lines to be configured as Open-Drain).

The user is also allowed to map internally both CAN modules on the same pins P4.5 and P4.6. In this way, P4.4 and P4.7 may be used either as general purpose I/O lines, or used for I^2C interface. This is possible by setting bit CANPAR of XMISC register. To access this register it is necessary to set bit XMISCEN of XPERCON register and bit XPEN of SYSCON register.

Note: CAN Parallel mode is effective only if both CAN1 and CAN2 are enabled through the setting of bits CAN1EN and CAN2EN in XPERCON register. If CAN1 is disabled, CAN2 remains on P4.4/P4.7 even if bit CANPAR is set.

XMISC (EB46h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–												VREG OFF	CAN CK2	CAN PAR	ADC MUX
–												RW	RW	RW	RW

Bit	Function
ADCMUX	**Port1L ADC Channels Enable** '0': Analog inputs on port P5.y can be converted (default configuration) '1': Analog inputs on port P1.z can be converted. Only 8 channels can be managed
CANPAR	**CAN Parallel Mode Selection** '0': CAN2 is mapped on P4.4/P4.7, while CAN1 is mapped on P4.5/P4.6 '1': CAN1 and CAN2 are mapped in parallel on P4.5/P4.6. This is effective only if both CAN1 and CAN2 are enabled through setting of bits CAN1EN and CAN2EN in XPERCON register. If CAN1 is disabled, CAN2 remains on P4.4/P4.7 even if bit CANPAR is set.
CANCK2	**CAN Clock divider by 2 disable** '0': Clock provided to CAN modules is CPU clock divided by 2 (mandatory when f_{CPU} is higher than 40 MHz) '1': Clock provided to CAN modules is directly CPU clock
VREGOFF	**Main Voltage Regulator disable for Power Down mode** '0': Default value after reset and when Power Down is not used. '1': On-chip Main Regulator is turned off when Power Down mode is entered.

21.3.1 Configuration examples

The following figures show different configuration examples, where the two CAN controllers of the ST10F276 are working on the same CAN bus or on different CAN busses.

Wired-or connections to a CAN bus use open drain outputs as described above. A wired-or structure can be used for on-board data exchange between two or more controller devices via one signal line. As no CAN transceiver is used in this case, the maximum wire length is very limited (<< 1 m) and noise conditions must be considered.

Finally, when one bus only is interfaced, the parallel mode for the two on-chip CAN modules allows to double the buffer capability, and to save two pins for other functionalities. The receive lines are internally tied together, while the transmit lines from the two modules are logically ANDed on the single pin: this is done to assign to the pin the active value driven by one of the two (for CAN protocol logic level '1' is the recessive state, so the non-transmitting CAN module, allows the other to drive the pin).

Note that after reset, the port pin are in high impedance, so an external pull-up is always needed to grant the recessive level on Tx lines.

Figure 172. Connection to single CAN bus via separate CAN transceivers

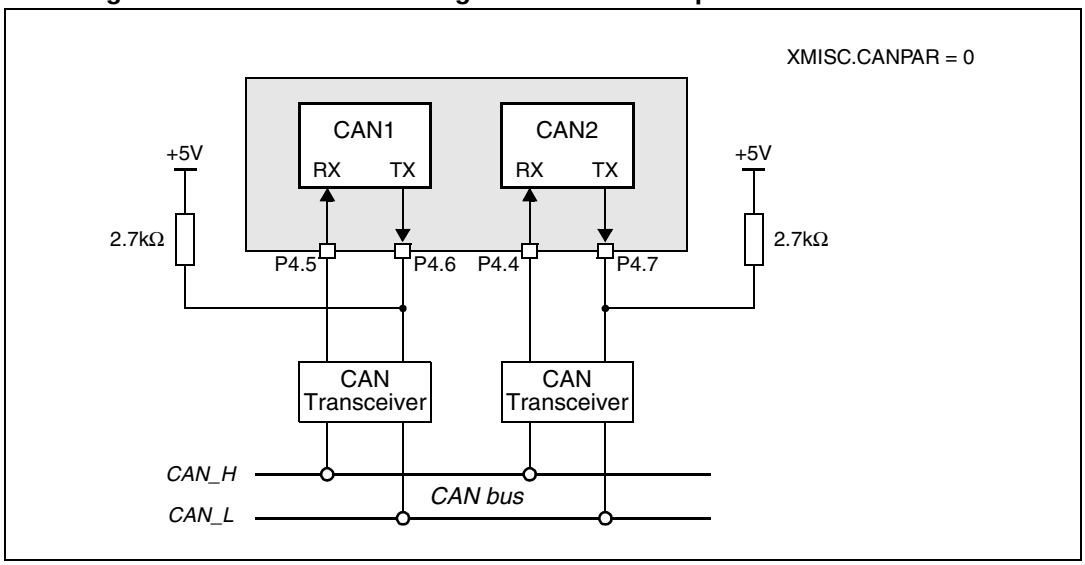


Figure 173. Connection to single CAN bus via one common transceiver

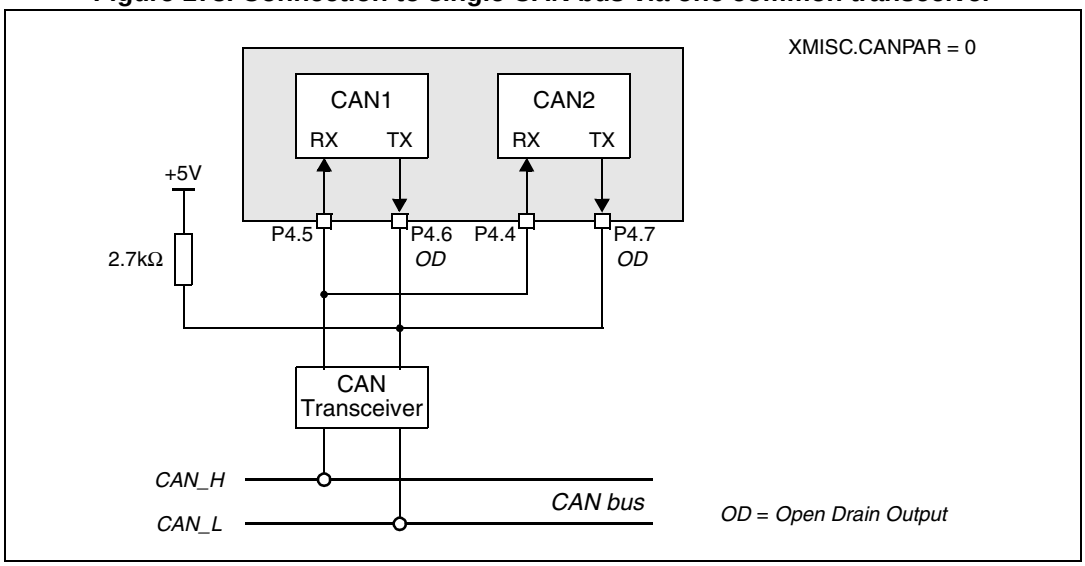


Figure 174. Connection to two different CAN buses (e.g. for gateway application)

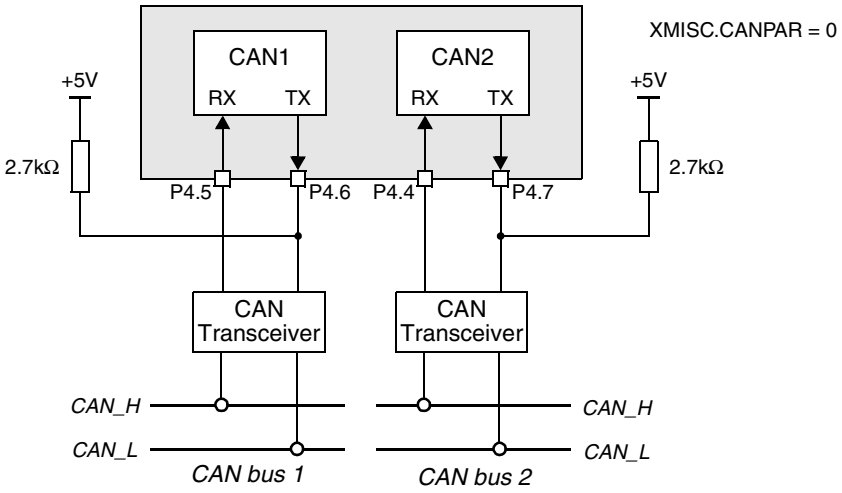
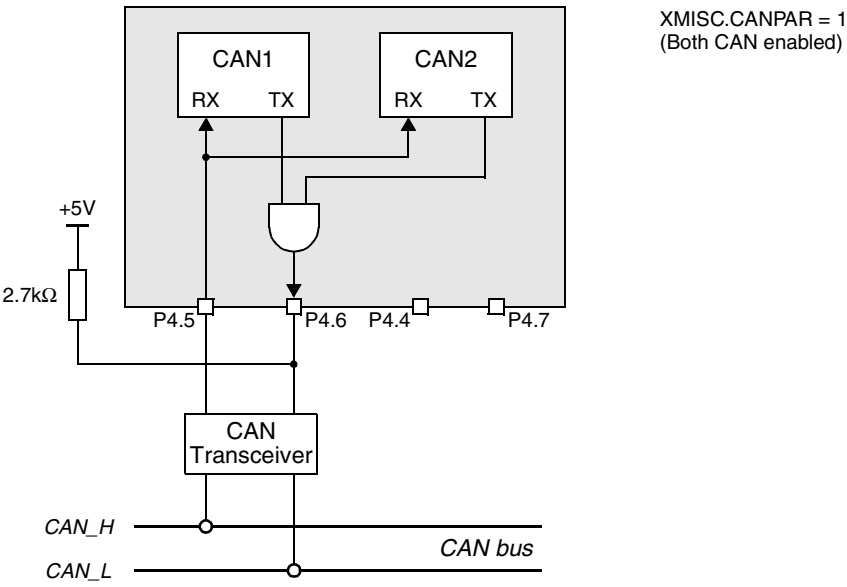


Figure 175. Connection to one CAN bus with internal parallel mode enabled



Note: P4.4 and P4.7 when not used as CAN functions can be used as general purpose I/O while they cannot be used as external bus address lines. Refer to *Section 6.6.1: Alternate functions of Port4* for more details.

21.4 Clock Prescaling

In the register XMISC there is also a bit (CANCK2) to modify the clock frequency driving both the CAN modules. For architectural limitations of the CAN module, when the CPU frequency is higher than 40 MHz, it is recommended to provide the CPU clock divided by 2 to each CAN module. 20 MHz is sufficient for CAN module to produce the maximum

baudrate defined by the protocol standard. On the other hand, the CPU frequency can be reduced down to 8 MHz: providing the CAN module directly with the CPU clock disabling the prescaler factor, it is still possible to obtain the maximum CAN speed (1Mbaud).

After reset the prescaler is enabled, so CPU clock is divided by 2 and then provided to the CAN modules: according to the system clock frequency, the application can disable the prescaler to obtain the required baudrate.

Refer to previous *Section 21.4: Clock Prescaling* for the description of register XMISC.

21.5 CAN module: functional overview

The C-CAN consists of the components (see *Figure 176*) CAN Core, Message RAM, Message Handler, Control Registers, and Module Interface.

The CAN Core performs communication according to the CAN protocol version 2.0 part A and B. The bitrate can be programmed to values up to 1 Mbit/s depending on the used technology. For the connection to the physical layer additional external transceiver hardware is required.

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.

The register set of the C-CAN can be accessed directly by the CPU via the module interface. These registers are used to control/configure the CAN Core and the Message Handler and to access the Message RAM.

The C-CAN implements the following features:

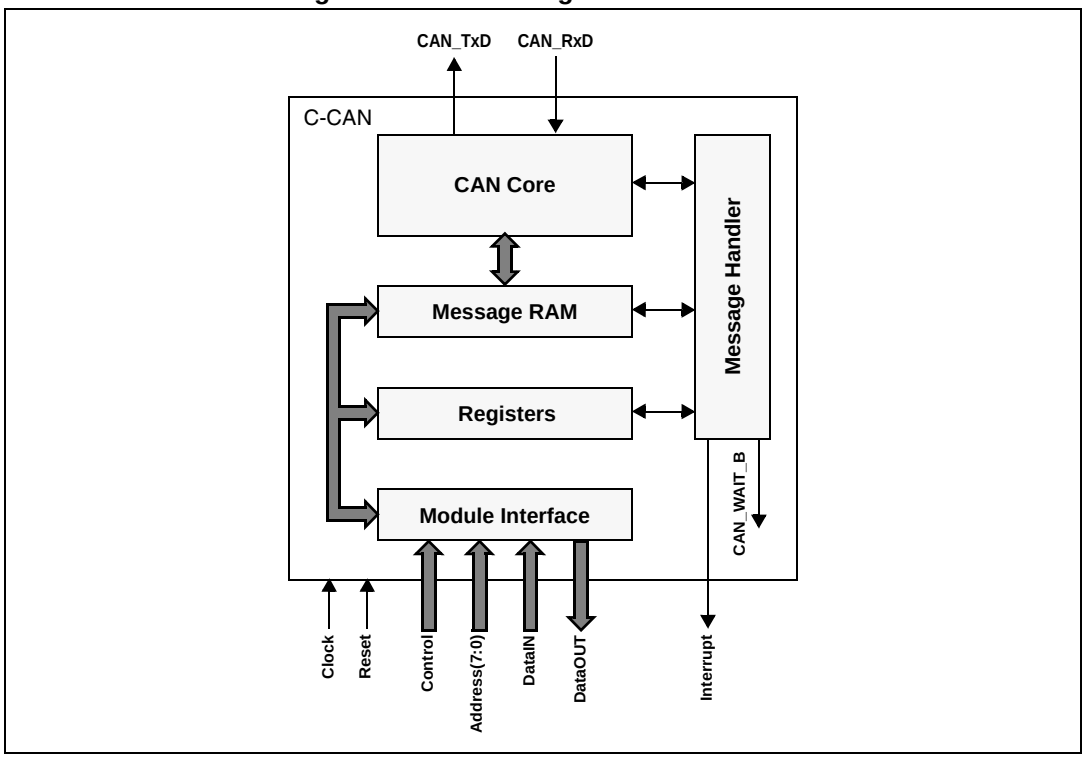
- Supports CAN protocol version 2.0 part A and B
- Bitrates up to 1 Mbit/s
- 32 Message Objects
- Each Message Object has its own identifier mask
- Programmable FIFO mode (concatenation of Message Objects)
- Maskable interrupt
- Disabled Automatic Retransmission mode for Time Triggered CAN applications
- Programmable loop-back mode for self-test operation

21.6 Block diagram

The module consists of the following functional blocks (see *Figure 176*):

- **CAN Core**: CAN Protocol Controller and Rx/Tx Shift Register for serial/parallel conversion of messages.
- **Message RAM:** Stores Message Objects and Identifier Masks.
- **Registers:** All registers used to control and to configure the C-CAN module.
- **Message Handler:** State Machine that controls the data transfer between the Rx/Tx Shift Register of the CAN Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers.

Figure 176. Block diagram of the C-CAN



21.7 Operating modes

21.7.1 Software initialization

The software initialization is started by setting the bit **Init** in the CAN Control Register, either by software or by a hardware reset, or by going *Bus_Off*.

While **Init** is set, all message transfer from and to the CAN bus is stopped, the status of the CAN bus output **CAN_TxD** is *recessive* (HIGH). The counters of the Error Management Logic are unchanged. Setting **Init** does not change any configuration register.

To initialize the CAN Controller, the CPU has to set up the Bit Timing Register and each Message Object. If a Message Object is not needed, it is sufficient to set its **MsgVal** bit to not valid. Otherwise, the whole Message Object has to be initialized.

Access to the Bit Timing Register and to the BRP Extension Register for the configuration of the bit timing is enabled when both bits **Init** and **CCE** in the CAN Control Register are set.

Resetting **Init** (by CPU only) finishes the software initialization. Afterwards the Bit Stream Processor BSP (see *Section 21.9.10: Configuration of the bit timing*) synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive *recessive* bits (≡ *Bus Idle*) before it can take part in bus activities and starts the message transfer.

The initialization of the Message Objects is independent of **Init** and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer.

To change the configuration of a Message Object during normal operation, the CPU has to start by setting **MsgVal** to not valid. When the configuration is completed, **MsgVal** is set to valid again.

21.7.2 CAN message transfer

Once the C-CAN is initialized and **Init** is reset to zero, the CAN Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes is stored into the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to "don't care" may be overwritten in the Message Object.

The CPU may read or write each message any time via the Interface Registers, the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits set up during configuration) exists for the message, only the data bytes are updated and then **TxRqst** bit with **NewDat** bit are set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, they are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

21.7.3 Disabled automatic re-transmission

According to the CAN Specification (see ISO11898, 6.3.3 Recovery Management), the C-CAN provides means for automatic re-transmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, this means for automatic re-transmission is enabled. It can be disabled to enable the C-CAN to work within a Time Triggered CAN (TTCAN, see ISO11898-1) environment.

The Disabled Automatic Retransmission mode is enabled by programming bit **DAR** in the CAN Control Register to *one*. In this operation mode the programmer has to consider the different behavior of bits **TxRqst** and **NewDat** in the Control Registers of the Message Buffers:

- When a transmission starts bit **TxRqst** of the respective Message Buffer is reset, while bit **NewDat** remains set.
- When the transmission completed successfully bit **NewDat** is reset.

When a transmission failed (lost arbitration or error) bit **NewDat** remains set. To restart the transmission the CPU has to set **TxRqst** back to *one*.

21.7.4 Test mode

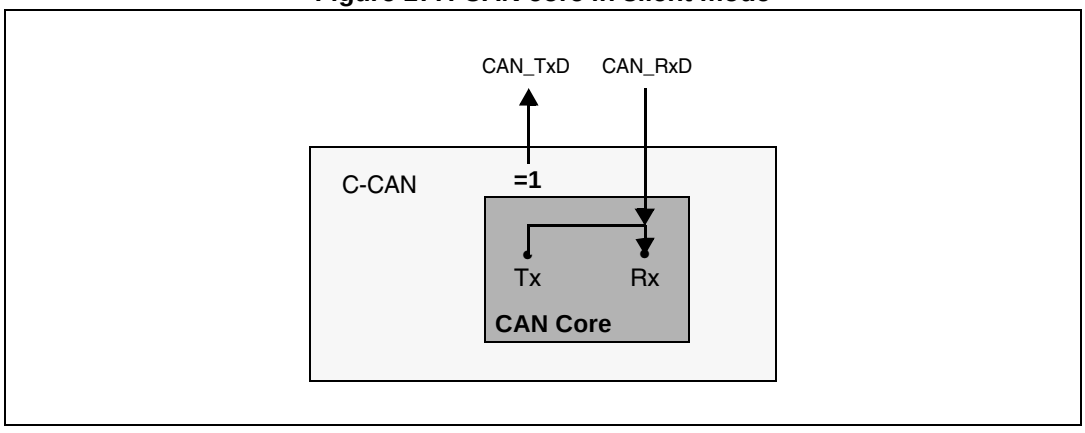
The Test Mode is entered by setting bit **Test** in the CAN Control Register to *one*. In Test Mode the bits **Tx1**, **Tx0**, **LBack**, **Silent** and **Basic** in the Test Register are writable. Bit **Rx** monitors the state of pin **CAN_RxD** and therefore is only readable. All Test Register functions are disabled when bit Test is reset to zero.

21.7.5 Silent mode

The CAN Core can be set in Silent Mode by programming the Test Register bit **Silent** to *one*.

In Silent Mode, the C-CAN is able to receive valid data frames and valid remote frames, but it sends only *recessive* bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a *dominant* bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyze the traffic on a CAN bus without affecting it by the transmission of *dominant* bits (Acknowledge Bits, Error Frames). *Figure 177* shows the connection of signals **CAN_TxD** and **CAN_RxD** to the CAN Core in Silent Mode.

Figure 177. CAN core in silent mode



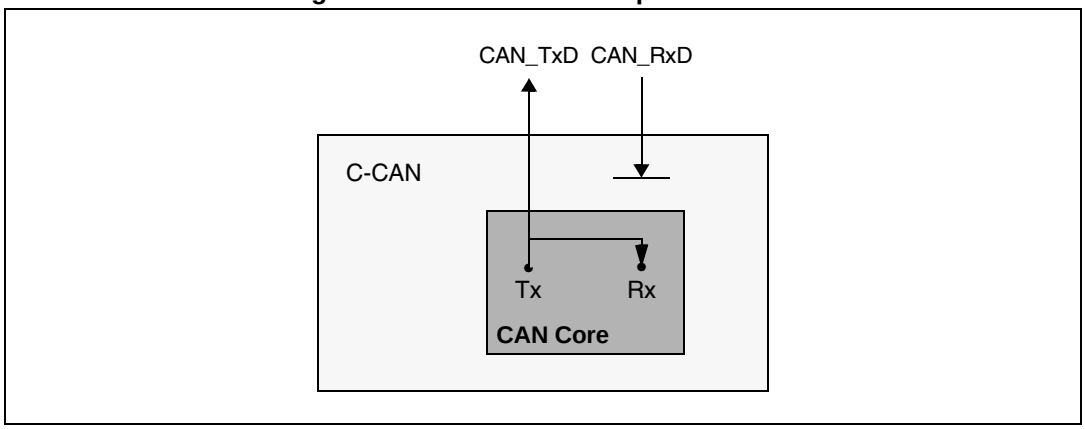
In ISO 11898-1, the Silent Mode is called the Bus Monitoring Mode.

21.7.6 Loop back mode

The CAN Core can be set in Loop Back Mode by programming the Test Register bit **LBack** to *one*. In Loop Back Mode, the CAN Core treats its own transmitted messages as received

messages and stores them (if they pass acceptance filtering) into a Receive Buffer. *Figure 178* shows the connection of signals **CAN_TxD** and **CAN_RxD** to the CAN Core in Loop Back Mode.

Figure 178. CAN core in loop back mode

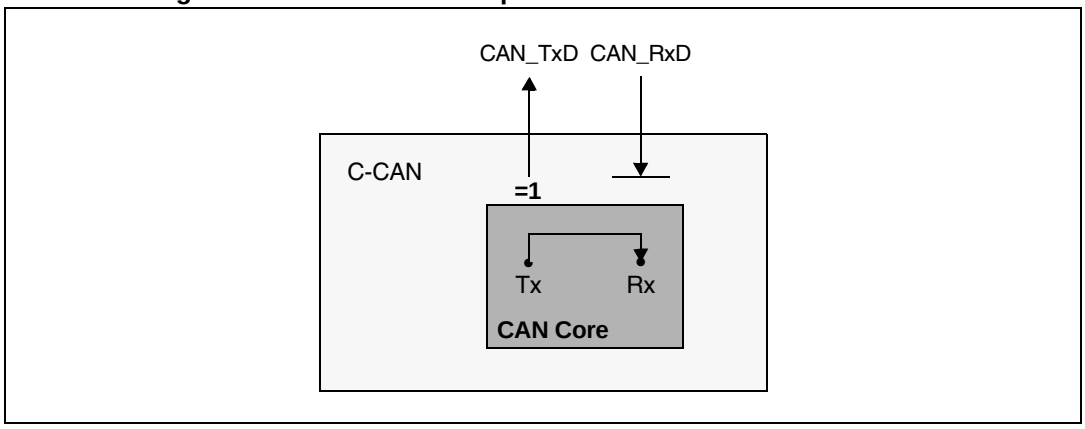


This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode the CAN Core performs an internal feedback from its Tx output to its Rx input. The actual value of the **CAN_RxD** input pin is disregarded by the CAN Core. The transmitted messages can be monitored at the **CAN_TxD** pin.

21.7.7 Loop back combined with silent mode

It is also possible to combine Loop Back Mode and Silent Mode by programming bits **LBack** and **Silent** to *one* at the same time. This mode can be used for a “Hot Selftest”, meaning the C-CAN can be tested without affecting a running CAN system connected to the pins **CAN_TxD** and **CAN_RxD**. In this mode the **CAN_RxD** pin is disconnected from the CAN Core and the **CAN_TxD** pin is held *recessive*. *Figure 179* shows the connection of signals **CAN_TxD** and **CAN_RxD** to the CAN Core in case of the combination of Loop Back Mode with Silent Mode.

Figure 179. CAN core in loop back combined with silent mode



21.7.8 Basic mode

The CAN Core can be set in Basic Mode by programming the Test Register bit **Basic** to *one*. In this mode the C-CAN module runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers is requested by writing the **Busy** bit of the IF1 Command Request Register to *one*. The IF1 Registers are locked while the **Busy** bit is set. The **Busy** bit indicates that the transmission is pending.

As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has completed, the **Busy** bit is reset and the locked IF1 Registers are released.

A pending transmission can be aborted at any time by resetting the **Busy** bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the Busy bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering.

Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the **Busy** bit of the IF2 Command Request Register to *one*, the contents of the shift register is stored into the IF2 Registers.

In Basic Mode the evaluation of all Message Object related control and status bits and of the control bits of the IFx Command Mask Registers is turned off. The message number of the Command request registers is not evaluated. The **NewDat** and **MsgLst** bits of the IF2 Message Control Register retain their function, **DLC(3:0)** will show the received **DLC**, the other control bits will be read as *zero*.

In Basic Mode the ready output **CAN_WAIT_B** is disabled (always *one*).

21.7.9 Software control of pin CAN_TxD

Four output functions are available for the CAN transmit pin **CAN_TxD**. Additionally to its default function – the serial data output – it can drive the CAN Sample Point signal to monitor CAN Core's bit timing and it can drive constant dominant or recessive values. The last two functions, combined with the readable CAN receive pin **CAN_RxD**, can be used to check the CAN bus' physical layer.

The output mode of pin **CAN_TxD** is selected by programming the Test Register bits **Tx1** and **Tx0** as described in *Section : Test register on page 428*.

The three test functions for pin **CAN_TxD** interfere with all CAN protocol functions. **CAN_TxD** must be left in its default function when CAN message transfer or any of the test modes Loop Back Mode, Silent Mode, or Basic Mode are selected.

21.8 Programmer's model

Each C-CAN module allocates an address space of 256 bytes. The registers are organized as 16-bit registers, with the high byte at the odd address and the low byte at the even address.

The two sets of interface registers (IF1 and IF2) control the CPU access to the Message RAM. They buffer the data to be transferred to and from the RAM, avoiding conflicts between CPU accesses and message reception/transmission.

Table 60. C-CAN register memory space summary

Address	Name	Reset value	Note
CAN Base + 0x00	CAN Control Register	0x0001	
CAN Base + 0x02	Status Register	0x0000	
CAN Base + 0x04	Error Counter	0x0000	read only
CAN Base + 0x06	Bit Timing Register	0x2301	write enabled by **CCE**
CAN Base + 0x08	Interrupt Register	0x0000	read only
CAN Base + 0x0A	Test Register	0x00 & 0br0000000 (1)	write enabled by **Test**
CAN Base + 0x0C	BRP Extension Register	0x0000	write enabled by **CCE**
CAN Base + 0x0E	— reserved	—(2)	
CAN Base + 0x10	IF1 Command Request	0x0001	
CAN Base + 0x12	IF1 Command Mask	0x0000	
CAN Base + 0x14	IF1 Mask 1	0xFFFF	
CAN Base + 0x16	IF1 Mask 2	0xFFFF	
CAN Base + 0x18	IF1 Arbitration 1	0x0000	
CAN Base + 0x1A	IF1 Arbitration 2	0x0000	
CAN Base + 0x1C	IF1 Message Control	0x0000	
CAN Base + 0x1E	IF1 Data A 1	0x0000	
CAN Base + 0x20	IF1 Data A 2	0x0000	
CAN Base + 0x22	IF1 Data B 1	0x0000	
CAN Base + 0x24	IF1 Data B 2	0x0000	
CAN Base + 0x28 - 0x3E	— reserved	— (2)	
CAN Base + 0x40 - 0x54	IF2 Registers	see note (3)	same as IF1 Registers
CAN Base + 0x56 - 0x7E	— reserved	— (2)	
CAN Base + 0x80	Transmission Request 1	0x0000	read only
CAN Base + 0x82	Transmission Request 2	0x0000	read only
CAN Base + 0x84 - 0x8E	— reserved	— (2)	
CAN Base + 0x90	New Data 1	0x0000	read only
CAN Base + 0x92	New Data 2	0x0000	read only
CAN Base + 0x94 - 0x9E	— reserved	— (2)	
CAN Base + 0xA0	Interrupt Pending 1	0x0000	read only
CAN Base + 0xA2	Interrupt Pending 2	0x0000	read only
CAN Base + 0xA4 - 0xAE	— reserved	— (2)	

Table 60. C-CAN register memory space summary

Address	Name	Reset value	Note
CAN Base + 0xB0	Message Valid 1	0x0000	read only
CAN Base + 0xB2	Message Valid 2	0x0000	read only
CAN Base + 0xB4 - 0xBE	— reserved	— (2)	

1. r signifies the actual value of the CAN_RxD pin.
2. Reserved bits are read as '0' except for IFx Mask 2 Register where they are read as '1'.
3. The two sets of Message Interface Registers - IF1 and IF2 - have identical functions.

21.8.1 Hardware reset description

After hardware reset, the registers of the C-CAN hold the values described in *Table 60*.

Additionally the *busoff* state is reset and the output **CAN_TxD** is set to *recessive* (HIGH). The value 0x0001 (**Init** = '1') in the CAN Control Register enables the software initialization. The C-CAN does not influence the CAN bus until the CPU resets **Init** to '0'.

The data stored in the Message RAM is not affected by a hardware reset. After Power-On, the contents of the Message RAM is undefined.

21.8.2 CAN protocol related registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

CAN control register

CAN1CR (EF00h) XBUS Reset Value: 0001h
CAN2CR (EE00h) XBUS Reset Value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	Test	CCE	DAR	-	EIE	SIE	IE	Init
								RW	RW	RW		RW	RW	RW	RW

Bit	Function
Init	**Initialization** '0': Normal Operation. '1': Initialization is started.
IE	**Module Interrupt Enable** '0': Disabled - Module Interrupt **IRQ_B** is always HIGH. '1': Enabled - Interrupts will set **IRQ_B** to LOW. **IRQ_B** remains LOW until all pending interrupts are processed.
SIE	**Status Change Interrupt Enable** '0': Disabled - No Status Change Interrupt will be generated. '1': Enabled - An interrupt will be generated when a message transfer is successfully completed or a CAN bus error is detected.

Bit	Function
EIE	**Error Interrupt Enable** '0': Disabled - No Error Status Interrupt will be generated. '1': Enabled - A change in the bits **BOff** or **EWarn** in the Status Register will generate an interrupt.
DAR	**Disable Automatic Retransmission** '0': Automatic Retransmission of disturbed messages enabled. '1': Automatic Retransmission disabled.
CCE	**Configuration Change Enable** '0': The CPU has no write access to the Bit Timing Register. '1': The CPU has write access to the Bit Timing Register (while **Init** = *one*).
Test	**Test Mode enable** '0': Normal Operation. '1': Test mode.

*Note: The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting bit **Init**. If the device goes busoff, it will set bit **Init** of its own accord, stopping all bus activities. Once bit **Init** has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129 * 11 consecutive recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.*

During the waiting time after the resetting of bit **Init**, each time a sequence of 11 *recessive* bits has been monitored, a **Bit0Error** code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus is stuck at *dominant* or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.

Status Register

CAN1SR (EF02h) XBUS Reset Value: 0000h
CAN2SR (EE02h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**BOff**	**EWarn**	**EPass**	**RxOk**	**TxOk**	**LEC**		
								R	R	R	RW	RW	RW		

Bit	Function
LEC	**Last Error Code** (Type of the last error to occur on the CAN bus) '000': No Error. '001': **Stuff Error:** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed. '010': **Form Error:** A fixed format part of a received frame has the wrong format. '011': **AckError:** The message this CAN Core transmitted was not acknowledged by another node. '100': **Bit1Error:** During the transmission of a message (with the exception of the arbitration field), the device wanted to send a *recessive* level (bit of logical value '1'), but the monitored bus value was *dominant*. '101': **Bit0Error:** During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), the device wanted to send a *dominant* level (data or identifier bit logical value '0'), but the monitored Bus value was *recessive*. During *busoff* recovery this status is set each time a sequence of 11 *recessive* bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continuously disturbed). '110': **CRCError:** The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data. '111': unused: When the LEC shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC.
TxOk	**Transmitted a Message Successfully** '0': Since this bit was reset by the CPU, no message has been successfully transmitted. This bit is never reset by the CAN Core. '1': Since this bit was last reset by the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted.
RxOk	**Received a Message Successfully** '0': Since this bit was last reset by the CPU, no message has been successfully received. This bit is never reset by the CAN Core. '1': Since this bit was last reset (to zero) by the CPU, a message has been successfully received (independent of the result of acceptance filtering).
EPass	**Error Passive** '0': The CAN Core is *error active*. '1': The CAN Core is in the *error passive* state as defined in the CAN Specification.
EWarn	**Warning Status** '0': Both error counters are below the error warning limit of 96. '1': At least one of the error counters in the Error Management Logic has reached the error warning limit of 96.
BOff	**Busoff Status** '0': The CAN module is not busoff. '1': The CAN module is in busoff state.

The **LEC** field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to '0h' when a message has been transferred (reception or transmission) without error. The unused code '7h' may be written by the CPU to check for updates.

Status interrupts

A Status Interrupt is generated by bits **BOff** and **EWarn** (Error Interrupt) or by **RxOk**, **TxOk**, and **LEC** (Status Change Interrupt) assumed that the corresponding enable bits in the CAN Control Register are set. A change of bit **EPass** or a write to **RxOk**, **TxOk**, or **LEC** will never generate a Status Interrupt.

Reading the Status Register will clear the Status Interrupt value (8000h) in the Interrupt Register, if it is pending.

Error counter

CAN1EC (EF04h) XBUS Reset Value: 0000h
CAN2EC (EE04h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RP	**REC(6:0)**							**TEC(7:0)**							
R	R							R							

Bit	Function
TEC(7:0)	Transmit Error Counter Actual state of the Transmit Error Counter. Values between 0 and 255.
REC(6:0)	Receive Error Counter Actual state of the Receive Error Counter. Values between 0 and 127.
RP	Receive Error Passive '0': The Receive Error Counter is below the *error passive* level. '1': The Receive Error Counter has reached the *error passive* level as defined in the CAN Specification.

Bit timing register

CAN1BTR (EF06h) XBUS Reset Value: 2301h
CAN2BTR (EE06h) XBUS Reset Value: 2301h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	TSeg2			TSeg1				SJW		BRP					
	RW			RW				RW		RW					

Bit	Function
BRP	**Baud Rate Prescaler** Value by which the CPU clock frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the Baud Rate Prescaler are 01h-3Fh (0…63). The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.
SJW	**(Re)Synchronization Jump Width** Valid programmed values are 0h-3h (0...3). The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bit	Function
TSeg1	**Time segment before the sample point** Valid values for TSeg1 are 01h-0Fh (1…15). The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.
TSeg2	**Time segment after the sample point** Valid values for TSeg2 are 0h-7h (0…7). The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

*Note: With a CAN module clock of 8 MHz, the reset value of 0x2301 configures the C-CAN for a bitrate of 500 Kbit/s. The registers are only writable if bits **CCE** and **Init** in the CAN Control Register are set.*

Test register

CAN1TR (EF0Ah) XBUS Reset Value: 00x0h
CAN2TR (EE0Ah) XBUS Reset Value: 00x0h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	Rx	Tx(1:0)		LBack	Silent	Basic	-	
								RW	RW		RW	RW	RW		

Bit	Function
Basic	**Basic Mode** ’0’: Basic Mode disabled. ’1’: IF1 Registers used as Tx Buffer, IF2 Registers used as Rx Buffer.
Silent	**Silent Mode** ’0’: Normal operation. ’1’: The module is in Silent Mode.
LBack	**Loop Back Mode** ’0’: Loop Back Mode is disabled. ’1’: Loop Back Mode is enabled.
Tx(1:0)	**CAN_TxD pin control** ’00’: Reset value, CAN_TxD is controlled by the CAN Core. ’01’: Sample Point can be monitored at CAN_TxD pin. ’10’: CAN_TxD pin drives a dominant (‘0’) value. ’11’: CAN_TxD pin drives a recessive (‘1’) value.
Rx	**Actual CAN_RxD pin value monitor** ’0’: The CAN bus is dominant (CAN_RxD = ‘0’). ’1’: The CAN bus is recessive (CAN_RxD = ‘1’).

Write access to the Test Register is enabled by setting bit **Test** in the CAN Control Register. The different test functions may be combined, but **Tx(1:0)** ≠ “00” disturbs message transfer.

BRP extension register

CAN1BRPER (EF0Ch) XBUS Reset Value: 0000h
CAN2BRPER (EE0Ch) XBUS Reset Value: 0000h

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

Bit	Function
BRPE	**Baud Rate Prescaler Extension** By programming **BRPE** (0h-Fh) the Baud Rate Prescaler can be extended to values up to 1023. The actual interpretation by the hardware is that one more than the value programmed by **BRPE** (MSBs) and **BRP** (LSBs) is used.

21.8.3 Message interface register sets

There are two sets of Interface Registers which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object (see *Message object in the message memory*) or parts of the Message Object may be transferred between the Message RAM and the IFx Message Buffer registers (see *IFx message buffer registers*) in one single transfer.

The function of the two interface register sets is identical (except for test mode **Basic**). They can be used the way that one set of registers is used for data transfer **to** the Message RAM while the other set of registers is used for the data transfer **from** the Message RAM, allowing both processes to be interrupted by each other. *Table 61* gives an overview of the two Interface Register sets.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

Table 61. IF1 and IF2 message interface register sets

Address	IF1 register set	Address	IF2 register set
CAN Base + 0x10	IF1 Command Request	CAN Base + 0x40	IF2 Command Request
CAN Base + 0x12	IF1 Command Mask	CAN Base + 0x42	IF2 Command Mask
CAN Base + 0x14	IF1 Mask 1	CAN Base + 0x44	IF2 Mask 1
CAN Base + 0x16	IF1 Mask 2	CAN Base + 0x46	IF2 Mask 2
CAN Base + 0x18	IF1 Arbitration 1	CAN Base + 0x48	IF2 Arbitration 1
CAN Base + 0x1A	IF1 Arbitration 2	CAN Base + 0x4A	IF2 Arbitration 2
CAN Base + 0x1C	IF1 Message Control	CAN Base + 0x4C	IF2 Message Control
CAN Base + 0x1E	IF1 Data A 1	CAN Base + 0x4E	IF2 Data A 1
CAN Base + 0x20	IF1 Data A 2	CAN Base + 0x50	IF2 Data A 2

Table 61. IF1 and IF2 message interface register sets

Address	IF1 register set	Address	IF2 register set
CAN Base + 0x22	IF1 Data B 1	CAN Base + 0x52	IF2 Data B 1
CAN Base + 0x24	IF1 Data B 2	CAN Base + 0x54	IF2 Data B 2

IFx command request registers

A message transfer is started as soon as the CPU has written the message number to the Command Request Register. With this write operation the **Busy** bit is automatically set to '1'. After a wait time of 3 to 6 CAN clock periods, the transfer between the Interface Register and the Message RAM has completed and the **Busy** bit is set back to *zero*.

CAN1IF1CR (EF10h) XBUS Reset Value: 0001h
CAN2IF1CR (EE10h) XBUS Reset Value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Busy	-	-	-	-	-	-	-	-	-	Message Number					
R										RW					

CAN1IF2CR (EF40h) XBUS Reset Value: 0001h
CAN2IF2CR (EE40h) XBUS Reset Value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Busy	-	-	-	-	-	-	-	-	-	Message Number					
R										RW					

Bit	Function
Message Number	**Message Number** '00h': Not a valid Message Number, interpreted as *20h*. '01h': Valid Message Number, the Message Object in the RAM is selected for data transfer. '02h': Valid Message Number, the Message Object in the RAM is selected for data transfer. '20h': Valid Message Number, the Message Object in the RAM is selected for data transfer. '21h': Not a valid Message Number, interpreted as *01h*. '22h': Not a valid Message Number, interpreted as *02h*. '3Fh': Not a valid Message Number, interpreted as *1Fh.*
Busy	**Busy Flag** '0': Reset to *zero* when read/write action has finished. '1': Set to *one* when writing to the IFx Command Request Register.

*Note: When a **Message Number** that is not valid is written into the Command Request Register, the **Message Number** will be transformed into a valid value and that Message Object will be transferred.*

IFx command mask registers

The control bits of the IFx Command Mask Register specify the transfer direction and select which of the IFx Message Buffer Registers are source or target of the data transfer. The bits of IFx Command Mask Register have different functions depending on the transfer direction (Write or Read), defined through WR/RD bit of the register itself.

CAN1IF1CM (EF12h) XBUS Reset Value: 0000h
CAN2IF1CM (EE12h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	WR/RD	Mask	Arb	Control	ClrInt Pnd	TxRqst/ NewDat	Data A	Data B
								RW	RW	RW	RW	RW	RW	RW	RW

CAN1IF2CM (EF42h) XBUS Reset Value: 0000h
CAN2IF2CM (EE42h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	WR/RD	Mask	Arb	Control	ClrInt Pnd	TxRqst/ NewDat	Data A	Data B
								RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
WR/RD	**Write / Read** '0': Read: Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers. '1': Write: Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register.

Direction Write:

Bit	Function
Data B	**Access Data Bytes 4-7** '0': Data Bytes 4-7 unchanged. '1': Transfer Data Bytes 4-7 to Message Object.
Data A	**Access Data Bytes 0-3** '0': Data Bytes 0-3 unchanged. '1': Transfer Data Bytes 0-3 to Message Object.
TxRqst/NewDat	**Access Transmission Request Bit** '0': TxRqst bit unchanged. '1': Set TxRqst bit. **Note:** If a transmission is requested by programming bit TxRqst/NewDat in the IFx Command Mask Register, bit TxRqst in the IFx Message Control Register will be ignored.
ClrIntPnd	**Clear Interrupt Pending Bit** When writing to a Message Object, this bit is ignored.

Bit	Function
Control	**Access Control Bits** '0': Control Bits unchanged. '1': Transfer Control Bits to Message Object.
Arb	**Access Arbitration Bits** '0': Arbitration bits unchanged. '1': Transfer **Identifier** + **Dir** + **Xtd** + **MsgVal** to Message Object.
Mask	**Access Mask Bits** '0': Mask bits unchanged. '1': Transfer **Identifier Mask** + **MDir** + **MXtd** to Message Object.

Direction Read:

Bit	Function
Data B	**Access Data Bytes 4-7** '0': Data Bytes 4-7 unchanged. '1': Transfer Data Bytes 4-7 to IFx Message Buffer Register.
Data A	**Access Data Bytes 0-3** '0': Data Bytes 0-3 unchanged. '1': Transfer Data Bytes 0-3 to IFx Message Buffer Register.
TxRqst/NewDat	**Access Transmission Request Bit** '0': NewDat bit remains unchanged. '1': Clear NewDat bit in the Message Object. A read access to a Message Object can be combined with the reset of the control bits IntPnd and NewDat. The values of these bits transferred to the IFx Message Control Register always reflect the status before resetting these bits.
ClrIntPnd	**Clear Interrupt Pending Bit** '0': IntPnd bit remains unchanged. '1': clear IntPnd bit in the Message Object.
Control	**Access Control Bits** '0': Control Bits unchanged. '1': Transfer Control Bits to IFx Message Buffer Register.
Arb	**Access Arbitration Bits** '0': Arbitration bits unchanged. '1': Transfer **Identifier** + **Dir** + **Xtd** + **MsgVal** to IFx Message Buffer Register.
Mask	**Access Mask Bits** '0': Mask bits unchanged. '1': Transfer **Identifier Mask** + **MDir** + **MXtd** to IFx Message Buffer Register.

IFx message buffer registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. The function of the Message Objects bits is described in *Message object in the message memory on page 438*.

IFx Mask Registers

CAN1IF1M1 (EF14h) XBUS Reset Value:FFFFh
CAN2IF1M1 (EE14h) XBUS Reset Value:FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Msk(15:0)															
RW															

CAN1IF1M2 (EF16h) XBUS Reset Value:FFFFh
CAN2IF1M2 (EE16h) XBUS Reset Value:FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MXtd	MDir	-	Msk(28:16)												
RW	RW		RW												

CAN1IF2M1 (EF44h) XBUS Reset Value:FFFFh
CAN2IF2M1 (EE44h) XBUS Reset Value:FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Msk(15:0)															
RW															

CAN1IF2M2 (EF46h) XBUS Reset Value:FFFFh
CAN2IF2M2 (EE46h) XBUS Reset Value:FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MXtd	MDir	-	Msk(28:16)												
RW	RW		RW												

Bit	Function
Msk(28:0)	**Identifier Mask** Msk(28:18): Identifier Mask Standard Message. Msk(28:0): Identifier Mask Extended Message. '0': The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering. '1': The corresponding identifier bit is used for acceptance filtering.
MDir	**Mask Message Direction** '0': The message direction bit (RTR) has no effect on the acceptance filtering. '1': The message direction bit (RTR) is used for acceptance filtering.
MXtd	**Mask Extended Identifier** '0': The extended identifier bit (IDE) has no effect on the acceptance filtering. '1': The extended identifier bit (IDE) is used for acceptance filtering.

IFx arbitration registers

CAN1IF1A1 (EF18h) XBUS Reset Value: 0000h
CAN2IF1A1 (EE18h) XBUS Reset Value: 0000h

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

CAN1IF1A2 (EF1Ah) XBUS Reset Value: 0000h
CAN2IF1A2 (EE1Ah) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal	Xtd	Dir	ID(28:16)												
RW	RW	RW	RW												

CAN1IF2A1 (EF48h) XBUS Reset Value: 0000h
CAN2IF2A1 (EE48h) XBUS Reset Value: 0000h

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

CAN1IF2A2 (EF4Ah) XBUS Reset Value: 0000h
CAN2IF2A2 (EE4Ah) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal	Xtd	Dir	ID(28:16)												
RW	RW	RW	RW												

Bit	Function
ID(28:0)	**Message Identifier** ID(28:18): Identifier for Standard Message. ID(28:0): Identifier for Extended Message.
Dir	**Message Direction** '0': Direction = Receive: on TxRqst, 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. '1': Direction = Transmit: on TxRqst, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TxRqst bit of this Message Object is set (if RmtEn = *one*).

Bit	Function
Xtd	**Extended Identifier** '0': The standard Identifier (11 bit) will be used for this Message Object. '1': The extended Identifier (29 bit) will be used for this Message Object.
MsgVal	**Message Valid** The CPU must reset the MsgVal bit of all unused Messages Objects during the initialization before it resets bit Init in the CAN Control Register. This bit must be reset if the message is no longer required, or if the identifier, the control bits Xtd, Dir or the Data Length Code DLC(3:0) are modified. '0': The Message Object is ignored by the Message Handler. '1': The Message Object is configured and should be considered by the Message Handler.

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 matching identifier and Direction = receive (Data Frame) or Direction = transmit (Remote Frame). Extended frames can be stored only in Message Objects with **Xtd** = *one*, standard frames in Message Objects with **Xtd** = *zero*. For identifier matching, the corresponding mask has to be considered. If the identifier of 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. For details see *Table 21.9: CAN application on page 444*.

IFx message control register

CAN1IF1MC (EF1Ch) XBUS Reset Value: 0000h
CAN2IF1MC (EE1Ch) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	-			DLC(3:0)			
RW	RW	RW	RW	RW	RW	RW	RW	RW				RW			

CAN1IF2MC (EF4Ch) XBUS Reset Value: 0000h
CAN2IF2MC (EE4Ch) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	-			DLC(3:0)			
RW	RW	RW	RW	RW	RW	RW	RW	RW				RW			

Bit	Function
DLC(3:0)	**Data Length Code** '0000': Data Frame has 0 data byte. '0001': Data Frame has 1 data byte. '1000': Data Frame has 8 data bytes. '1001': Data Frame has 8 data bytes. '1111': Data Frame has 8 data bytes. **Note:** The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.
EoB	**End of Buffer** '0': FIFO Operation: If MsgVal is set, this messageObject stores the next Message. '1': FIFO Operation: Last Message Object of a FIFO Buffer. **Note:** This bit is used to concatenate two ore more Message Objects (up to 32) to build a FIFO Buffer. For normal operation this bit must always be set to *one*.
TxRqst	**Transmit Request** '0': This Message Object is not waiting for transmission. '1': The transmission of this Message Object is requested and is not yet done.
RmtEn	**Remote Enable** '0': At the reception of a Remote Frame, TxRqst is left unchanged. '1': At the reception of a Remote Frame, TxRqst is set.
RxIE	**Receive Interrupt Enable** '0': No interrupt is generated after the successful reception of a frame. '1': An interrupt is generated after a successful reception of a frame (this interrupt may be masked by IE bit in the CAN Control Register).
TxIE	**Transmit Interrupt Enable** '0': No interrupt is generated after the successful transmission of a frame. '1': An interrupt is generated after a successful transmission of a frame (this interrupt may be masked by IE bit in the CAN Control Register).
UMask	**Use Identifier Mask** '0': Identifier Mask ignored. '1': Use identifier Mask.
IntPnd	**Interrupt Pending** '0': No interrupt was generated by this message object since last time the CPU has cleared this flag. '1': This message object has generated an interrupt.

Bit	Function
MsgLst	**Message Lost** (only valid for Message Objects with direction = *receive*) '0': No message lost since last time this bit was reset by the CPU. '1': The Message Handler stored a new message into this object when NewDat was still set, the CPU has lost a message.
NewDat	**New Data** '0': No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU. '1': The Message Handler or the CPU has written new data into the data portion of this Message Object.

IFx Data A and Data B registers

The data bytes of CAN messages are stored in the IFx Message Buffer Registers (or Data Registers) in the order shown in the next table.

CAN1IF1DA1 (EF1Eh) XBUS Reset Value: 0000h
CAN2IF1DA1 (EE1Eh) XBUS Reset Value: 0000h

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

CAN1IF1DA2 (EF20h) XBUS Reset Value: 0000h
CAN2IF1DA2 (EE20h) XBUS Reset Value: 0000h

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

CAN1IF1DB1 (EF22h) XBUS Reset Value: 0000h
CAN2IF1DB1 (EE22h) XBUS Reset Value: 0000h

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

CAN1IF1DB2 (EF24h) XBUS Reset Value: 0000h
CAN2IF1DB2 (EE24h) XBUS Reset Value: 0000h

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

CAN1IF2DA1 (EF4Eh) XBUS Reset Value: 0000h
CAN2IF2DA1 (EE4Eh) XBUS Reset Value: 0000h

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

CAN1IF2DA2 (EF50h) XBUS Reset Value: 0000h
CAN2IF2DA2 (EE50h) XBUS Reset Value: 0000h

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

CAN1IF2DB1 (EF52h) XBUS Reset Value: 0000h
CAN2IF2DB1 (EE52h) XBUS Reset Value: 0000h

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

CAN1IF2DB2 (EF54h) XBUS Reset Value: 0000h
CAN2IF2DB2 (EE54h) XBUS Reset Value: 0000h

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

In a CAN Data Frame, Data(0) is the first, Data(7) is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

Message object in the message memory

There are 32 Message Objects in the Message RAM. To avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission, the CPU cannot directly access the Message Objects, these accesses are handled via the IFx Interface Registers.

Next diagram gives an overview of the two structures of a Message Object in the Message Memory.

Message object												
UMask	Msk(28:0)	MXtd	MDir	EoB	NewDat		MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
MsgVal	ID(28:0)	Xtd	Dir	DLC(3:0)	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

Bit	Function
MsgVal	**Message Valid** '0': The Message Object is ignored by the Message Handler. '1': The Message Object is configured and should be considered by the Message Handler. **Note:** The CPU must reset the MsgVal bit of all unused Messages Objects during the initialization before it resets bit Init in the CAN Control Register. This bit must also be reset before the identifier ID(28:0), the control bits Xtd, Dir, or the Data Length Code DLC(3:0) are modified, or if the Messages Object is no longer required.
UMask	**Use Acceptance Mask** '0': Mask ignored. '1': Use Mask (Msk28-0, MXtd, and MDir) for acceptance filtering. If the UMask bit is set to one, the Message Object's mask bits have to be programmed during initialization of the Message Object before MsgVal is set to one.
ID(28:0)	**Message Identifier** ID(28:18): 11-bit Identifier (Standard Frame). ID(28:0): 29-bit Identifier (Extended Frame).
Msk(28:0)	**Identifier Mask** '0': The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering. '1': The corresponding identifier bit is used for acceptance filtering.
Xtd	**Extended Identifier** '0': The standard Identifier (11 bit) will be used for this Message Object. '1': The extended Identifier (29 bit) will be used for this Message Object.
MXtd	**Mask Extended Identifier** '0': The extended identifier bit (IDE) has no effect on the acceptance filtering. '1': The extended identifier bit (IDE) is used for acceptance filtering. **Note:** When 11-bit ("standard") Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits ID28 to ID18. For acceptance filtering, only these bits together with mask bits Msk28 to Msk18 are considered.
Dir	**Message Direction** '0': Direction = Receive: on TxRqst, 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. '1': Direction = Transmit: on TxRqst, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TxRqst bit of this Message Object is set (if RmtEn = *one*).

Bit	Function
MDir	**Mask Message Direction** '0': The message direction bit (Dir) has no effect on the acceptance filtering. '1': The message direction bit (Dir) is used for acceptance filtering. The Arbitration Registers ID(28:0), Xtd, and Dir are used to define the identifier and type of outgoing messages and are used (together with the mask registers Msk(28:0), MXtd, and MDir) for acceptance filtering of incoming messages. A received message is stored into the valid Message Object with matching identifier and Direction = *receive* (Data Frame) or Direction = *transmit* (Remote Frame). Extended frames can be stored only in Message Objects with Xtd = *one*, standard frames in Message Objects with Xtd = *zero*. 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. For details see *Acceptance filtering of received messages on page 447*.
EoB	**End of Buffer** '0': Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer. '1': Single Message Object or last Message Object of a FIFO Buffer. **Note:** This bit is used to concatenate two ore more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects (not belonging to a FIFO Buffer) this bit must always be set to one. For details on the concatenation of Message Objects see *Section 21.9.7: Configuration of a FIFO buffer on page 449*.
NewDat	**New Data** '0': No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU. '1': The Message Handler or the CPU has written new data into the data portion of this Message Object.
MsgLst	**Message Lost** (only valid for Message Objects with direction = *receive*) '0': No message lost since last time this bit was reset by the CPU. '1': The Message Handler stored a new message into this object when NewDat was still set, the CPU has lost a message.
RxIE	**Receive Interrupt Enable** '0': IntPnd will be left unchanged after a successful reception of a frame. '1': IntPnd will be set after a successful reception of a frame.
TxIE	**Transmit Interrupt Enable** '0': IntPnd will be left unchanged after the successful transmission of a frame. '1': IntPnd will be set after a successful transmission of a frame.
IntPnd	**Interrupt Pending** '0': This message object is not the source of an interrupt. '1': This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.
RmtEn	**Remote Enable** '0': At the reception of a Remote Frame, TxRqst is left unchanged. '1': At the reception of a Remote Frame, TxRqst is set.

Bit	Function
TxRqst	**Transmit Request** '0': This Message Object is not waiting for transmission. '1': The transmission of this Message Object is requested and is not yet done.
DLC(3:0)	**Data Length Code** '0000': Data Frame has 0 data byte. '0001': Data Frame has 1 data byte. '1000': Data Frame has 8 data bytes. '1001': Data Frame has 8 data bytes. '1111': Data Frame has 8 data bytes. **Note:** The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.
Data 0 Data 1 Data 2 Data 3 Data 4 Data 5 Data 6 Data 7	1st data byte of a CAN Data Frame 2nd data byte of a CAN Data Frame 3rd data byte of a CAN Data Frame 4th data byte of a CAN Data Frame 5th data byte of a CAN Data Frame 6th data byte of a CAN Data Frame 7th data byte of a CAN Data Frame 8th data byte of a CAN Data Frame **Note:** Byte **Data 0** is the first data byte shifted into the shift register of the CAN Core during a reception, byte **Data 7** is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by **non specified values**.

21.8.4 Message handler registers

All Message Handler registers are read-only. Their contents (**TxRqst**, **NewDat**, **IntPnd**, and **MsgVal** bits of each Message Object and the Interrupt Identifier) is status information provided by the Message Handler FSM.

Interrupt register

CAN1IR (EF08h) XBUS Reset Value: 0000h
CAN2IR (EE08h) XBUS Reset Value: 0000h

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

Bit	Function
IntId(15:0)	**Interrupt identifier** (the number here indicates the source of the interrupt) '0000h': No interrupt is pending. '0001h': Message Object 1 caused the interrupt. '0020h': Message Object 32 caused the interrupt. '0021h': Unused. '7FFFh': Unused. '8000h': Status Interrupt. '8001h': Unused. 'FFFFh': Unused.

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If **IntId** is different from 0x0000 and **IE** is set, the interrupt line to the CPU, is active. The interrupt line remains active until **IntId** is back to value 0x0000 (the cause of the interrupt is reset) or until **IE** is reset.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's **IntPnd** bit. The Status Interrupt is cleared by reading the Status Register.

Transmission request registers

CAN1TR1 (EF80h) XBUS Reset Value: 0000h
CAN2TR1 (EE80h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TxRqst(16:1)															
R															

CAN1TR2 (EF82h) XBUS Reset Value: 0000h
CAN2TR2 (EE82h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TxRqst(32:17)															
R															

Bit	Function
TxRqst(32:1)	**Transmission Request Bits** (of all Message Objects) '0': This Message Object is not waiting for transmission. '1': The transmission of this Message Object is requested and is not yet done.

These registers hold the TxRqst bits of the 32 Message Objects. By reading out the TxRqst bits, the CPU can check for which Message Object a Transmission Request is pending. The TxRqst bit of a specific Message Object can be set/reset by the CPU via the IFx Message

Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

New data registers

CAN1ND1 (EF90h) XBUS Reset Value: 0000h
CAN2ND1 (EE90h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat(16:1)															
R															

CAN1ND2 (EF92h) XBUS Reset Value: 0000h
CAN2ND2 (EE92h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat(32:17)															
R															

Bit	Function
NewDat(32:1)	**New Data Bits** (of all Message Objects) '0': No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU. '1': The Message Handler or the CPU has written new data into the data portion of this Message Object.

These registers hold the **NewDat** bits of the 32 Message Objects. By reading out the **NewDat** bits, the CPU can check for which Message Object the data portion was updated. The **NewDat** bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

Interrupt pending registers

CAN1IP1 (EFA0h) XBUS Reset Value: 0000h
CAN2IP1 (EEA0h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IntPnd(16:1)															
R															

CAN1IP2 (EFA2h) XBUS Reset Value: 0000h
CAN2IP2 (EEA2h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IntPnd(32:17)															
R															

Bit	Function
IntPnd(32:1)	**Interrupt Pending Bits** (of all Message Objects) '0': This message object is not the source of an interrupt. '1': This message object is the source of an interrupt.

These registers hold the **IntPnd** bits of the 32 Message Objects. By reading out the **IntPnd** bits, the CPU can check for which Message Object an interrupt is pending. The **IntPnd** bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of **IntId** in the Interrupt Register.

Message valid registers

CAN1MV1 (EFB0h) XBUS Reset Value: 0000h
CAN2MV1 (EEB0h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal(16:1)															
R															

CAN1MV2 (EFB2h) XBUS Reset Value: 0000h
CAN2MV2 (EEB2h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal(32:17)															
R															

Bit	Function
MsgVal(32:1)	**Message Valid Bits** (of all Message Objects) '0': This Message Object is ignored by the Message Handler. '1': This Message Object is configured and should be considered by the Message Handler.

These registers hold the **MsgVal** bits of the 32 Message Objects. By reading out the **MsgVal** bits, the CPU can check which Message Object is valid. The **MsgVal** bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers.

21.9 CAN application

21.9.1 Management of message objects

The configuration of the Message Objects in the Message RAM will (with the exception of the bits **MsgVal**, **NewDat**, **IntPnd**, and **TxRqst**) not be affected by resetting the chip. All the Message Objects must be initialized by the CPU or they must be not valid (**MsgVal** = '0') and the bit timing must be configured before the CPU clears the **Init** bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data field of one of the two interface register sets to the desired values. By writing to the corresponding IFx Command Request Register, the IFx Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the **Init** bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN Core and the Message Handler State Machine control the C-CAN's internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, messages with pending transmission request are loaded into the CAN Core's Shift Register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFx Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

21.9.2 Message handler state machine

The Message Handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IFx Registers.

The Message Handler Finite State Machine (FSM) controls the following functions:

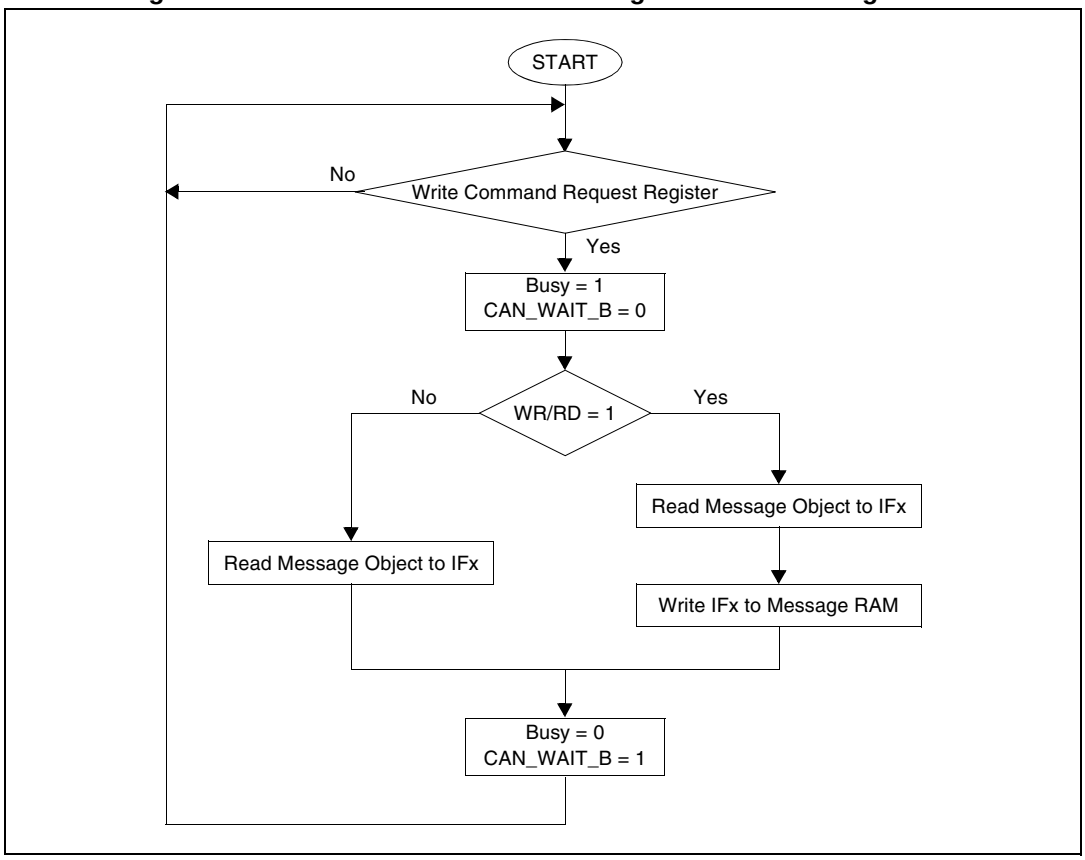
- Data Transfer from IFx Registers to the Message RAM
- Data Transfer from Message RAM to the IFx Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of **TxRqst** flags
- Handling of interrupts

Data Transfer from / to message RAM

When the CPU initiates a data transfer between the IFx Registers and Message RAM, the Message Handler sets the **Busy** bit in the respective Command Register to '1'. After the transfer has completed, the **Busy** bit is set back to '0' (see *Figure 180*).

The respective Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM, it is not possible to write single bits/bytes of one Message Object, it is always necessary to write a complete Message Object into the Message RAM. Therefore the data transfer from the IFx Registers to the Message RAM requires of a read-modify-write cycle. First, those parts of the Message Object that are not to be changes are read from the Message RAM, and then the complete contents of the Message Buffer Registers are written into the Message Object.

Figure 180. Data transfer between IFx registers and message RAM



After the partial write of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be set to the actual contents of the selected Message Object.

After the partial read of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be left unchanged.

Transmission of messages

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFx Registers and Message RAM, the **MsgVal** bits in the Message Valid Register and **TxRqst** bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The Message Object's **NewDat** bit is reset.

After a successful transmission and if no new data was written to the Message Object (**NewDat** = '0') since the start of the transmission, the **TxRqst** bit will be reset. If **TxIE** is set, **IntPnd** will be set after a successful transmission. If the C-CAN has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. If meanwhile the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

Acceptance filtering of received messages

When the arbitration and control field (Identifier + IDE + RTR + DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler FSM starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. Then the arbitration and mask fields (including **MsgVal**, **UMask**, **NewDat** and **EoB**) of Message Object 1 are loaded into the Acceptance Filtering unit and compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scanning is stopped and the Message Handler FSM proceeds depending on the type of frame (Data Frame or Remote Frame) received.

Reception of data frame

The Message Handler FSM stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but all arbitration bits and the Data Length Code are stored into the corresponding Message Object. This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NewDat bit is set to indicate that new data (not yet seen by the CPU) has been received. The CPU should reset NewDat bit when it reads the Message Object. If at the time of the reception the NewDat bit was already set, MsgLst is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RxIE bit is set, the IntPnd bit is set, causing the Interrupt Register to point to this Message Object.

The TxRqst bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

Reception of remote frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1. **Dir** = ‘1’ (direction = *transmit*), **RmtEn** = ‘1’, **UMask** = ‘1’ or ‘0’
   At the reception of a matching Remote Frame, the **TxRqst** bit of this Message Object is set. The rest of the Message Object remains unchanged.
2. **Dir** = ‘1’ (direction = *transmit*), **RmtEn** = ‘0’, **UMask** = ‘0’
   At the reception of a matching Remote Frame, the **TxRqst** bit of this Message Object remains unchanged; the Remote Frame is ignored.
3. **Dir** = ‘1’ (direction = *transmit*), **RmtEn** = ‘0’, **UMask** = ‘1’
   At the reception of a matching Remote Frame, the **TxRqst** bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored into the Message Object in the Message RAM and the **NewDat** bit of this Message Object is set. The data field of the Message Object remains unchanged; the Remote Frame is treated similar to a received Data Frame.

Receive / transmit priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority.

If more than one transmission request is pending, they are serviced according to the priority of the corresponding Message Object.

21.9.3 Configuration of a transmit object

The next diagram shows how a Transmit Object should be initialized.

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl	appl.	appl	1	1	0	0	0	appl	0	appl.	0

The Arbitration Registers (**ID(28:0)** and **Xtd** bit) are given by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to **ID(28:18)**, while **ID(17:0)** can then be disregarded.

If the **TxIE** bit is set, the **IntPnd** bit will be set after a successful transmission of the Message Object.

If the **RmtEn** bit is set, a matching received Remote Frame will cause the **TxRqst** bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Registers (**DLC(3:0)** and **Data(7:0)**) are given by the application, **TxRqst** and **RmtEn** may not be set before the data is valid.

The Mask Registers (**Msk(28:0)**, **UMask**, **MXtd**, and **MDir** bits) may be used (**UMask** = ‘1’) to allow groups of Remote Frames with similar identifiers to set the **TxRqst** bit. For details see *Reception of remote frame on page 447*, and handle with care. The **Dir** bit should not be masked.

21.9.4 Updating a transmit object

The CPU may update the data bytes of a Transmit Object any time via the IFx Interface registers, neither **MsgVal** nor **TxRqst** have to be reset before the update.

Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFx Data A Register or IFx Data B Register have to be valid before the content of that register is transferred to the Message Object. Either the CPU has to write all four bytes into the IFx Data Register or the Message Object is transferred to the IFx Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x0087h is written to the Command Mask Register and then the number of the Message Object is written to the Command Request Register, concurrently updating the data bytes and setting **TxRqst**.

To prevent the reset of **TxRqst** at the end of a transmission that may already be in progress while the data is updated, **NewDat** has to be set together with **TxRqst**. For details see *Transmission of messages on page 446*.

When **NewDat** is set together with **TxRqst**, **NewDat** bit will be reset as soon as the new transmission has started.

21.9.5 Configuration of a receive object

The next diagram shows how a Receive Object should be initialized.

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl	appl	appl	1	0	0	0	appl	0	0	0	0

The Arbitration Registers (**ID(28:0)** and **Xtd** bit) are given by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to **ID(28:18)**, while **ID(17:0)** can then be disregarded. When a Data Frame with an 11-bit Identifier is received, **ID(17:0)** will be reset to ‘0’.

If the **RxIE** bit is set, the **IntPnd** bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (**DLC(3:0)**) is given by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by **non specified values**.

The Mask Registers (**Msk(28:0)**, **UMask**, **MXtd** and **MDir** bits) may be used (**UMask** = ‘1’) to allow groups of Data Frames with similar identifiers to be accepted. For details see *Reception of data frame on page 447*. The **Dir** bit should not be masked in typical applications.

21.9.6 Handling of received messages

The CPU may read a received message any time via the IFx Interface registers, the data consistency is guaranteed by the Message Handler state machine.

Typically the CPU will write first 0x007Fh to the Command Mask Register and then the number of the Message Object to the Command Request Register. That combination will transfer the whole received message from the Message RAM into the Message Buffer Register. Additionally, the bits **NewDat** and **IntPnd** are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits show which of the matching messages has been received.

The actual value of **NewDat** shows whether a new message has been received since last time this Message Object was read. The actual value of **MsgLst** shows whether more than one message has been received since last time this Message Object was read. **MsgLst** will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the **TxRqst** bit of a receive object will cause the transmission of a Remote Frame with the receive object’s identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the **TxRqst** bit is automatically reset.

21.9.7 Configuration of a FIFO buffer

With the exception of the **EoB** bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object, see *Section 21.9.5 on page 448*.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the

implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The **EoB** bit of all Message Objects of a FIFO Buffer except the last have to be programmed to *zero*. The **EoB** bits of the last Message Object of a FIFO Buffer is set to *one*, configuring it as the **E**nd **o**f the **B**lock.

21.9.8 Reception of messages with FIFO buffers

Received messages with identifiers matching to a FIFO Buffer are stored into a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored into a Message Object of a FIFO Buffer the **NewDat** bit of this Message Object is set. By setting **NewDat** while **EoB** is *zero* the Message Object is locked for further write accesses by the Message Handler until the CPU has written the **NewDat** bit back to *zero*.

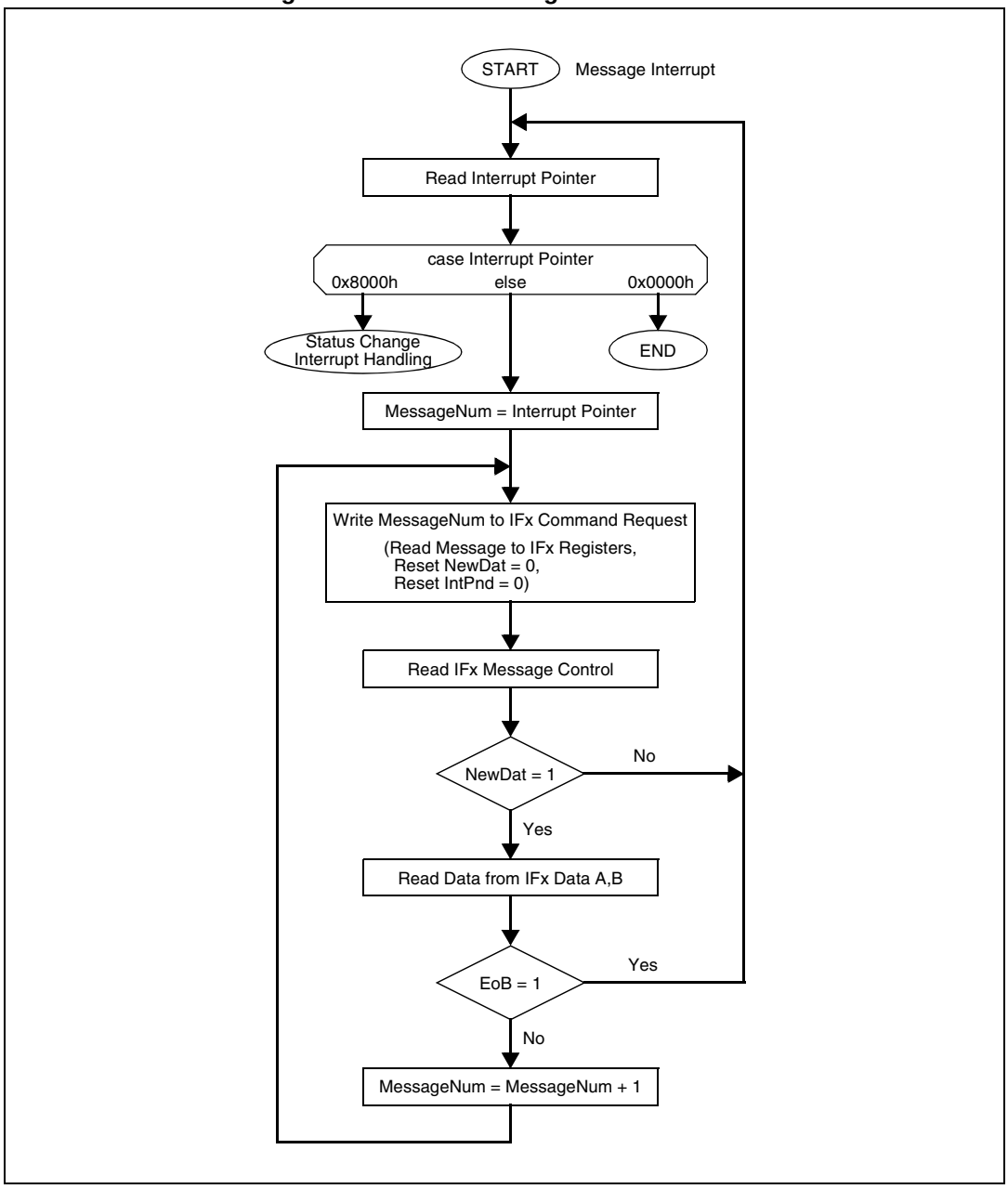
Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing **NewDat** to *zero*, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

Reading from a FIFO buffer

When the CPU transfers the contents of Message Object to the IFx Message Buffer registers by writing its number to the IFx Command Request Register, the corresponding Command Mask Register should be programmed the way that bits **NewDat** and **IntPnd** are reset to *zero* (**TxRqst/NewDat** = ‘1’ and **ClrIntPnd** = ‘1’). The values of these bits in the Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read out the Message Objects starting at the FIFO Object with the lowest message number. *Figure 181* shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.

Figure 181. CPU handling of a FIFO buffer



21.9.9 Handling of interrupts

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's IntPnd bit. The Status Interrupt is cleared by reading the Status Register.

The interrupt identifier **IntId** in the Interrupt Register indicates the cause of the interrupt. When no interrupt is pending, the register will hold the value *zero*. If the value of the Interrupt Register is different from *zero*, then there is an interrupt pending and, if **IE** is set, the interrupt line to the CPU, **IRQ_B**, is active. The interrupt line remains active until the Interrupt Register is back to value *zero* (the cause of the interrupt is reset) or until **IE** is reset.

The value 0x8000h indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the status bits **RxOk**, **TxOk** and **LEC**, but a write access of the CPU to the Status Register can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects, **IntId** points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Register may cause an interrupt (bits **EIE** and **SIE** in the CAN Control Register) and whether the interrupt line becomes active when the Interrupt Register is different from *zero* (bit **IE** in the CAN Control Register). The Interrupt Register will be updated even when **IE** is reset.

The CPU has two possibilities to follow the source of a message interrupt. First it can follow the **IntId** in the Interrupt Register and second it can poll the Interrupt Pending Register (see *Interrupt pending registers on page 443*).

An interrupt service routine reading the message that is the source of the interrupt may read the message and reset the Message Object's **IntPnd** at the same time (bit **ClrIntPnd** in the Command Mask Register). When **IntPnd** is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

21.9.10 Configuration of the bit timing

Even if minor errors in the configuration of the CAN bit timing do not result in immediate failure, the performance of a CAN network can be reduced significantly.

In many cases, the CAN bit synchronization will amend a faulty configuration of the CAN bit timing to such a degree that only occasionally an error frame is generated. In the case of arbitration however, when two or more CAN nodes simultaneously try to transmit a frame, a misplaced sample point may cause one of the transmitters to become error passive.

The analysis of such sporadic errors requires a detailed knowledge of the CAN bit synchronization inside a CAN node and of the CAN nodes' interaction on the CAN bus.

Bit time and bitrate

CAN supports bitrates in the range of lower than 1 Kbit/s up to 1000 Kbit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (that is, the reciprocal of the bitrate) can be configured individually for each CAN node, creating a common bitrate even though the CAN nodes' oscillator periods (f_{osc}) may be different.

The frequencies of these oscillators (or PLL's when used to generate the CAN clock starting from a reference obtained through a quartz oscillator) are not absolutely stable, small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specific oscillator tolerance range (df), the CAN nodes are able to compensate for the different bitrates by resynchronizing to the bit stream.

According to the CAN specification, the bit time is divided into four segments (see *Figure 182*): the Synchronization Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta (see *Table 62*). The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{sys} and the Baud Rate Prescaler (BRP): t_q = BRP / f_{sys}. The C-CAN's system clock f_{sys} is the frequency of its CAN module clock input (see f_{CPU}).

The Synchronization Segment Sync_Seg is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-)Synchronization Jump Width (SJW) defines how far a resynchronization may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

Figure 182. Bit timing

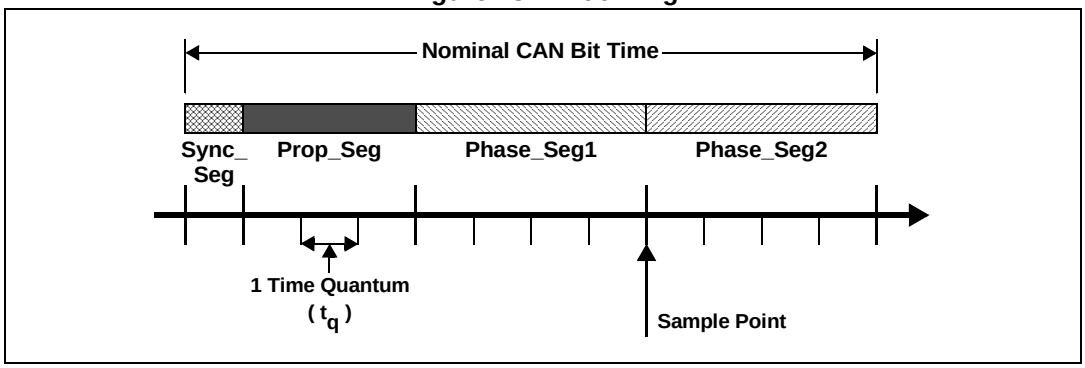


Table 62. Parameters of the CAN bit time

Parameter	Range	Remark
BRP BRP + BRPE	[1 .. 32] [1 .. 512]	defines the length of the time quantum t_q
Sync_Seg	1 t_q	fixed length, synchronization of bus input to system clock
Prop_Seg	[1 .. 8] t_q	compensates for the physical delay times
Phase_Seg1	[1 .. 8] t_q	may be lengthened temporarily by synchronization
Phase_Seg2	[1 .. 8] t_q	may be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	may not be longer than either Phase Buffer Segments
This table describes the minimum programmable ranges required by the CAN protocol		

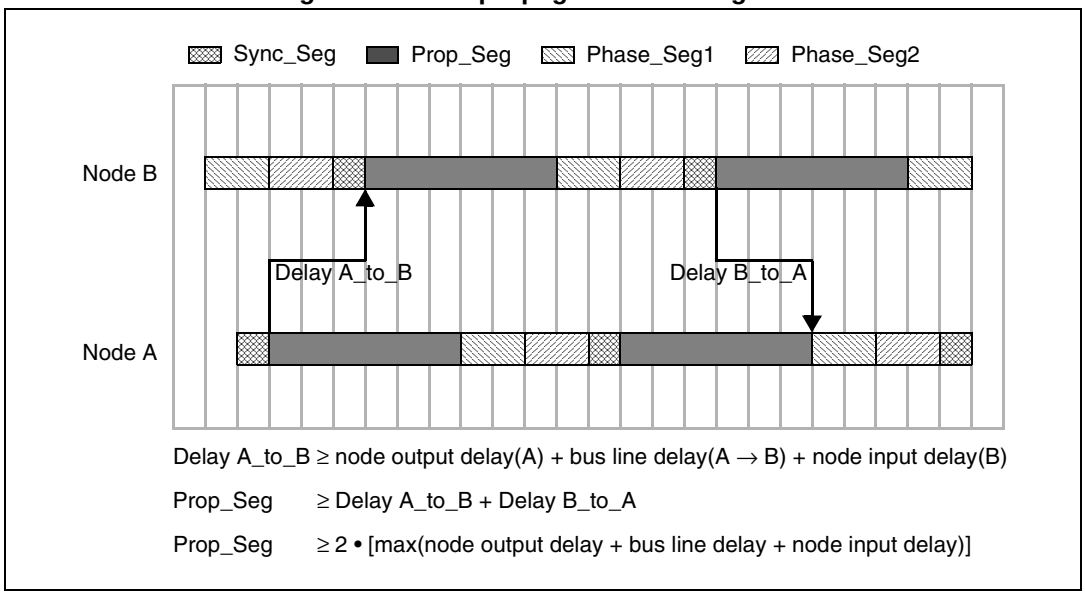
A given bitrate may be met by different bit time configurations, but for the proper function of the CAN network the physical delay times and the oscillator's tolerance range have to be considered.

Propagation time segment

This part of the bit time is used to compensate physical delay times within the network. These delay times consist of the signal propagation time on the bus and the internal delay time of the CAN nodes.

Any CAN node synchronized to the bit stream on the CAN bus will be out of phase with the transmitter of that bit stream, caused by the signal propagation time between the two nodes. The CAN protocol's non-destructive bit wise arbitration and the dominant acknowledge bit provided by receivers of CAN messages require that a CAN node transmitting a bit stream must also be able to receive dominant bits transmitted by other CAN nodes that are synchronized to that bit stream. The example in *Figure 183* shows the phase shift and propagation times between two CAN nodes.

Figure 183. The propagation time segment



In this example, both nodes A and B are transmitters performing an arbitration for the CAN bus. The node A has sent its Start of Frame bit less than one bit time earlier than node B, therefore node B has synchronized itself to the received edge from recessive to dominant. Since node B has received this edge delay(A_to_B) after it has been transmitted, B's bit timing segments are shifted with regard to A. Node B sends an identifier with higher priority and so it will win the arbitration at a specific identifier bit when it transmits a dominant bit while node A transmits a recessive bit. The dominant bit transmitted by node B will arrive at node A after the delay(B_to_A).

Due to oscillator (or PLL) tolerances, the actual position of node A's Sample Point can be anywhere inside the nominal range of node A's Phase Buffer Segments, so the bit transmitted by node B must arrive at node A before the start of Phase_Seg1. This condition defines the length of Prop_Seg.

If the edge from recessive to dominant transmitted by node B would arrive at node A after the start of Phase_Seg1, it could happen that node A samples a recessive bit instead of a dominant bit, resulting in a bit error and the destruction of the current frame by an error flag.

The error occurs only when two nodes arbitrate for the CAN bus that have oscillators (or PLL's) of opposite ends of the tolerance range and that are separated by a long bus line; this

is an example of a minor error in the bit timing configuration (Prop_Seg too short) that causes sporadic bus errors.

Some CAN implementations provide an optional 3 Sample Mode: the C-CAN does not. In this mode, the CAN bus input signal passes a digital low-pass filter, using three samples and a majority logic to determine the valid bit value. This results in an additional input delay of 1 t_q, requiring a longer Prop_Seg.

Phase buffer segments and synchronization

The Phase Buffer Segments (Phase_Seg1 and Phase_Seg2) and the Synchronization Jump Width (SJW) are used to compensate for the oscillator (or PLL) tolerance. The Phase Buffer Segments may be lengthened or shortened by synchronization.

Synchronizations occur on edges from recessive to dominant, their purpose is to control the distance between edges and Sample Points.

Edges are detected by sampling the actual bus level in each time quantum and comparing it with the bus level at the previous Sample Point. A synchronization may be done only if a recessive bit was sampled at the previous Sample Point and if the actual time quantum's bus level is dominant.

An edge is synchronous if it occurs inside of Sync_Seg, otherwise the distance between edge and the end of Sync_Seg is the edge phase error, measured in time quanta. If the edge occurs before Sync_Seg, the phase error is negative, else it is positive.

Two types of synchronization exist: Hard Synchronization and Resynchronization. A Hard Synchronization is done once at the start of a frame; inside a frame only Resynchronizations occur.

- **Hard synchronization**
  After a hard synchronization, the bit time is restarted with the end of Sync_Seg, regardless of the edge phase error. Thus hard synchronization forces the edge which has caused the hard synchronization to lie within the synchronization segment of the restarted bit time.
- **Bit resynchronization**
  Resynchronization leads to a shortening or lengthening of the bit time such that the position of the sample point is shifted with regard to the edge.
  When the phase error of the edge which causes Resynchronization is positive, Phase_Seg1 is lengthened. If the magnitude of the phase error is less than SJW, Phase_Seg1 is lengthened by the magnitude of the phase error, else it is lengthened by SJW.
  When the phase error of the edge which causes Resynchronization is negative, Phase_Seg2 is shortened. If the magnitude of the phase error is less than SJW, Phase_Seg2 is shortened by the magnitude of the phase error, else it is shortened by SJW.

When the magnitude of the phase error of the edge is less than or equal to the programmed value of SJW, the results of Hard Synchronization and Resynchronization are the same. If the magnitude of the phase error is larger than SJW, the Resynchronization cannot compensate the phase error completely, an error of (phase error - SJW) remains.

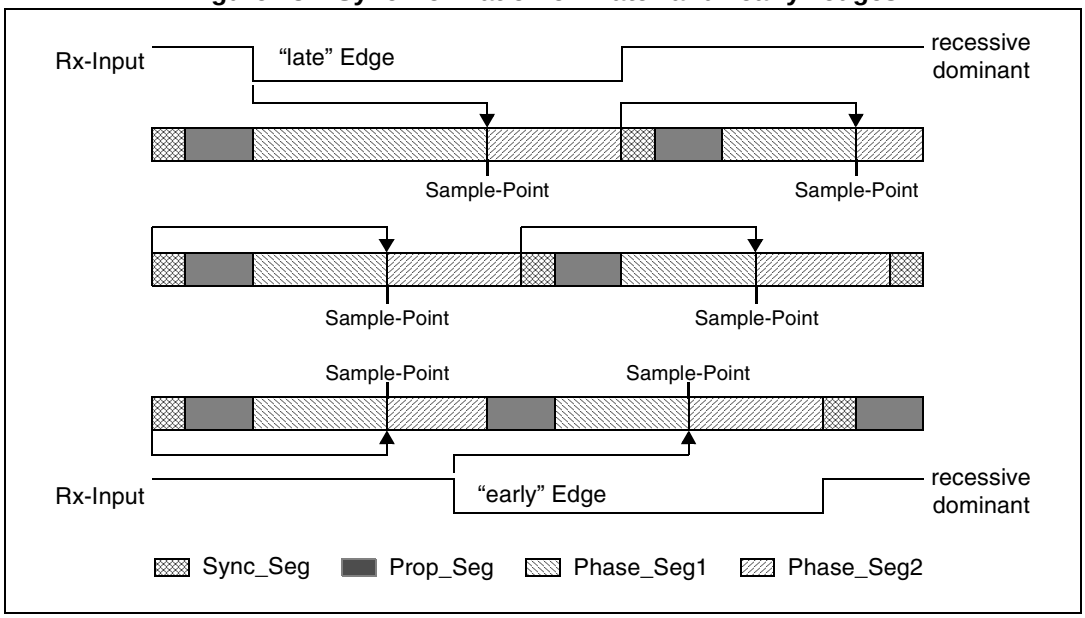
Only one synchronization may be done between two Sample Points. The Synchronizations maintain a minimum distance between edges and Sample Points, giving the bus level time to stabilize and filtering out spikes that are shorter than (Prop_Seg + Phase_Seg1).

Apart from noise spikes, most synchronizations are caused by arbitration. All nodes synchronize “hard” on the edge transmitted by the “leading” transceiver that started transmitting first, but due to propagation delay times, they cannot become ideally synchronized. The “leading” transmitter does not necessarily win the arbitration, therefore the receivers have to synchronize themselves to different transmitters that subsequently “take the lead” and that are differently synchronized to the previously “leading” transmitter. The same happens at the acknowledge field, where the transmitter and some of the receivers will have to synchronize to that receiver that “takes the lead” in the transmission of the dominant acknowledge bit.

Synchronizations after the end of the arbitration will be caused by oscillator tolerance, when the differences in the oscillator’s clock periods of transmitter and receivers sum up during the time between synchronizations (at most ten bits). These summarized differences may not be longer than the SJW, limiting the oscillator’s (or PLL) tolerance range.

The examples in *Figure 184* show how the Phase Buffer Segments are used to compensate for phase errors. There are three drawings of each two consecutive bit timings. The upper drawing shows the synchronization on a “late” edge, the lower drawing shows the synchronization on an “early” edge, and the middle drawing is the reference without synchronization.

Figure 184. Synchronization on “late” and “early” edges



In the first example an edge from recessive to dominant occurs at the end of Prop_Seg. The edge is “late” since it occurs after the Sync_Seg. Reacting to the “late” edge, Phase_Seg1 is lengthened so that the distance from the edge to the Sample Point is the same as it would have been from the Sync_Seg to the Sample Point if no edge had occurred. The phase error of this “late” edge is less than SJW, so it is fully compensated and the edge from dominant to recessive at the end of the bit, which is one nominal bit time long, occurs in the Sync_Seg.

In the second example an edge from recessive to dominant occurs during Phase_Seg2. The edge is “early” since it occurs before a Sync_Seg. Reacting to the “early” edge, Phase_Seg2 is shortened and Sync_Seg is omitted, so that the distance from the edge to the Sample Point is the same as it would have been from a Sync_Seg to the Sample Point if

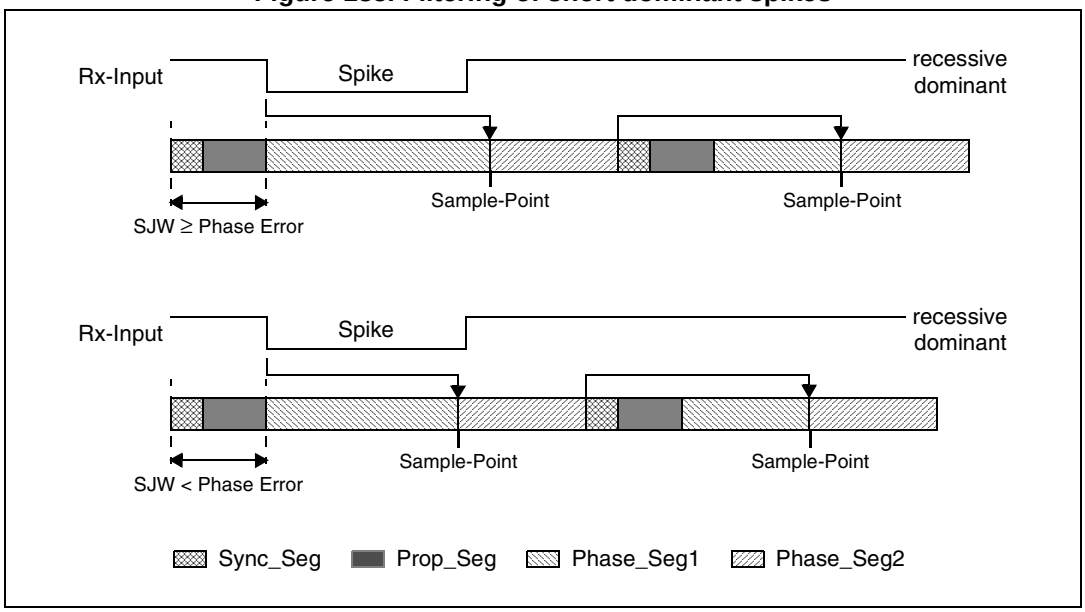
no edge had occurred. As in the previous example, the magnitude of this “early” edge’s phase error is less than SJW, so it is fully compensated.

The Phase Buffer Segments are lengthened or shortened temporarily only; at the next bit time, the segments return to their nominal programmed values.

In these examples, the bit timing is seen from the point of view of the CAN implementation’s state machine, where the bit time starts and ends at the Sample Points. The state machine omits Sync_Seg when synchronizing on an “early” edge because it cannot subsequently redefine that time quantum of Phase_Seg2 where the edge occurs to be the Sync_Seg.

The examples in *Figure 185* show how short dominant noise spikes are filtered by synchronizations. In both examples the spike starts at the end of Prop_Seg and has the length of (Prop_Seg + Phase_Seg1).

Figure 185. Filtering of short dominant spikes



In the first example, the Synchronization Jump Width is greater than or equal to the phase error of the spike’s edge from recessive to dominant. Therefore the Sample Point is shifted after the end of the spike; a recessive bus level is sampled.

In the second example, SJW is shorter than the phase error, so the Sample Point cannot be shifted far enough; the dominant spike is sampled as actual bus level.

System clock tolerance range

The CAN system clock for the different nodes in the network is typically derived from a different clock generator source. The actual CAN system clock frequency for each node (and consequently the actual bit time), is affected by a tolerance. In particular, for ST10F276 the CAN system clock is derived (prescaled) from the CPU clock, typically generated by the on-chip PLL multiplying the frequency of the main oscillator.

An effective communication requires that all CAN nodes in the network sample the correct value for each transmitted bit: also those nodes (typically at opposite ends of the network) with the largest propagation delay, and working with system clocks that are at opposite limits of the frequency tolerance, must be able to correctly receive and decode every message transmitted on the network.

Considering the effect of the system clock discrepancy between two CAN nodes, and supposing no bus errors is detected (due to, for instance, electrical disturbances), bit stuffing guarantees that, also in the worst case condition for the accumulation of phase error (during a normal communication), the maximum time between two re-synchronization edges is 10 bit periods (5 dominant bits followed by 5 recessive bits are always followed by a dominant bit).

Calling t_{BT} the CAN Bit Time, this maximum time t_J between two re-synchronization edges can be simply expressed as:

$$t_J = 10 \cdot t_{BT}$$

Then assuming the two CAN nodes with opposite system clock generator tolerance (considering the specified tolerance "df" valid for both the nodes in the network) for their respective system clocks, the accumulated phase error at the re-synchronization instant becomes:

$$\Delta t_J = (2 \cdot df) \cdot 10 \cdot t_{BT}$$

where df represents the system clock relative tolerance (f actual frequency, f_N nominal frequency):

$$df = \frac{|f - f_N|}{f_N}$$

This error must be compensated, therefore it must be less than the programmed (Re-)Synchronization Jump Width (SJW). Calling t_{SJW} the duration of the re-synchronization segment (programmable from 1 to 4 time quanta), the following condition can be written:

$$(2 \cdot df) \cdot 10 \cdot t_{BT} < t_{SJW}$$

This expression can be seen as a condition for the CAN system clock tolerance df:

$$df < \frac{t_{SJW}}{2 \cdot 10 \cdot t_{BT}}$$

Considering now that real systems typically operate in the presence of electrical disturbances, errors on the CAN bus may occurs. When an error is detected, an Error Flag is transmitted on the bus: if the error is just local, only the node which detected it transmits the Error Flag on the bus, while the other nodes simply receive the Error Flag and then transmit their own Error Flags as an echo. On the contrary, if the error is global, all nodes detect it within the same bit time, so they transmit their own Error Flags simultaneously. In this way, each node can recognize if the error is local or global simply by detecting whether there is an echo after its Error Flag. This is possible only if the node can properly sample the first bit after transmitting the Error Flag.

The Error Flag from an Error Active node is composed by 6 dominant bits; in the worst case condition of a bit stuffing error, up to other 6 dominant bits could be received before the Error Flag. It means that the first bit after the Error Flag is the 13th bit after the last synchronization: this bit, as already said, must be correctly sampled.

Again calling t_{BT} the CAN Bit Time, the maximum time t_S (with correct sampling) between two re-synchronization edges can be expressed as:

$$t_S = 13 \cdot t_{BT} - t_{PB2}$$

where t_{PB2} corresponds to the duration of Phase_Seg2 (PB = Phase Buffer).

Also in this case, assuming the two CAN nodes with opposite system clock generator tolerance (considering the specified tolerance "df" valid for both the nodes in the network) for their respective system clocks, the accumulated phase error at the re-synchronization instant becomes:

$$\Delta t_S = (2 \cdot df) \cdot (13 \cdot t_{BT} - t_{PB2})$$

For a correct sampling, the accumulated phase error must not lead the re-synchronization edge outside the interval Phase_seg1 + Phase_Seg2. This condition can be expressed as:

$$t_{PB1} < (2 \cdot df) \cdot (13 \cdot t_{BT} - t_{Seg2}) < t_{PB2}$$

Once again, this expression can be translated in a condition for the CAN system clock tolerance df:

$$df < \frac{\min(t_{PB1}, t_{PB2})}{2 \cdot (13 \cdot t_{BT} - t_{PB2})}$$

In conclusion, there are two conditions on CAN system clock tolerance both to be satisfied.

In case the CAN node generates its system clock through a PLL, the maximum allowed clock tolerance must be a function of the PLL jitter also: this will result in a more severe quality requirement for the oscillator (crystal or resonator).

The phase error introduced by the PLL jitter is a function of the number of clock periods: in particular the jitter increases with the clock period number till a saturation maximum value, which consists in the long term jitter. Refer to datasheet for more details about the PLL Electrical Characteristics.

Considering the PLL effect, the two expressions giving the phase error in the two conditions of introduced above, are modified as in the following:

$$\Delta t_J = 2 \cdot (df \cdot 10 \cdot t_{BT} + \delta_{PLL})$$

$$\Delta t_S = 2 \cdot [df \cdot (13 \cdot t_{BT} - t_{PB2}) + \delta_{PLL}]$$

where δ_{PLL} represents the absolute deviation introduced by the PLL jitter. In the two formulas the value of δ_{PLL} should be evaluated for different number of clock periods: for the first, the jitter correspondent to 10 bit time period must be considered, while for the second, the jitter correspondent to 13 bit time period must be considered. The number of clock periods should be computed taking into account the baudrate prescaler setting as well. Again, the factor 2, which multiplies the single CAN node phase deviation, is considered to take into account the worst case eventuality that the two communicating nodes are at the opposite limits of the specified frequency tolerance.

From two equations above, the new constraints for the CAN system clock tolerance can be translated in new quality requirements for the oscillator:

$$df < \frac{t_{SJW} - 2 \cdot \delta_{PLL}}{2 \cdot 10 \cdot t_{BT}}$$

$$df < \frac{\min(t_{PB1}, t_{PB2}) - 2 \cdot \delta_{PLL}}{2 \cdot (13 \cdot t_{BT} - t_{PB2})}$$

It is evident that the PLL jitter imposes a more stringent constraints on oscillator tolerance than what can be accepted when no PLL is used. ST10F276 PLL characteristics are such that the oscillator requirements are acceptably impacted by the jitter for the majority of the worst CAN bus network configurations.

The oscillator tolerance range was increased when the CAN protocol was developed from version 1.1 to version 1.2 (version 1.0 was never implemented in silicon). The option to synchronize on edges from dominant to recessive became obsolete, only edges from recessive to dominant are considered for synchronization. The protocol update to version 2.0 (A and B) had no influence on the oscillator tolerance.

It has to be considered that SJW may not be larger than the smaller of the Phase Buffer Segments and that the Propagation Time Segment limits the part of the bit time that may be used for the Phase Buffer Segments.

The combination Prop_Seg = 1 and Phase_Seg1 = Phase_Seg2 = SJW = 4 allows the largest possible frequency tolerance of 1.58% (in the absence of PLL jitter). This combination with a Propagation Time Segment of only 10% of the bit time is not suitable for short bit times; it can be used for bitrates of up to 125 Kbit/s (bit time = 8µs) with a bus length of 40m.

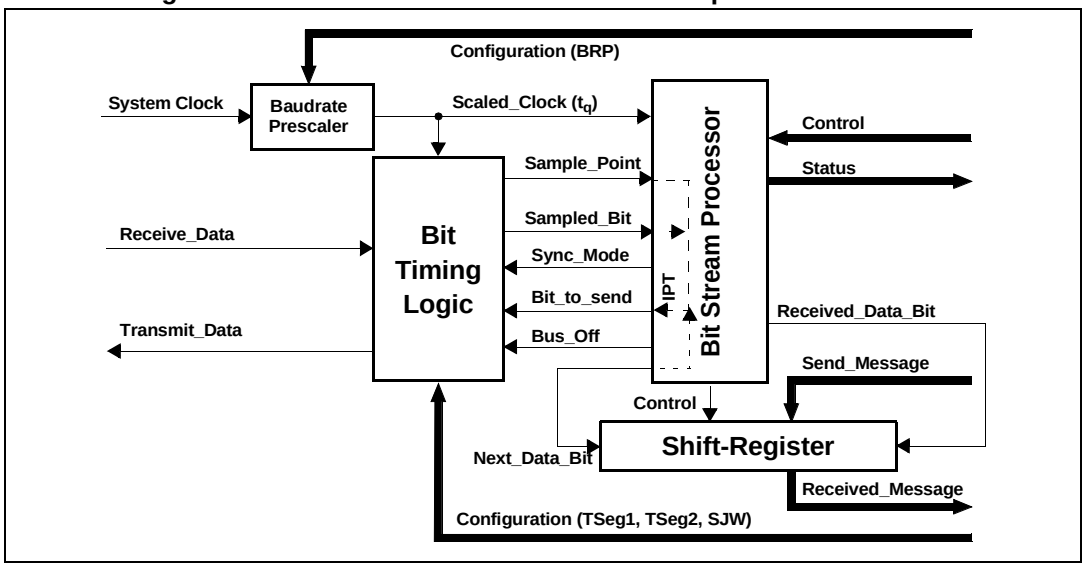
Configuration of the CAN protocol controller

In most CAN implementations and also in the C-CAN, the bit timing configuration is programmed in two register bytes. The sum of Prop_Seg and Phase_Seg1 (as TSeg1) is combined with Phase_Seg2 (as TSeg2) in one byte, SJW and BRP are combined in the other byte (see *Figure 186*).

In these bit timing registers (CANxBTR), the four components TSeg1, TSeg2, SJW, and BRP have to be programmed to a numerical value that is one less than its functional value; so instead of values in the range of [1...n], values in the range of [0...n-1] are programmed. That way, for example, SJW (functional range of [1...4]) is represented by only two bits.

Therefore the length of the bit time is (programmed values) [TSeg1 + TSeg2 + 3] t_q or (functional values) [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] t_q.

Figure 186. Structure of the CAN Core's CAN protocol controller



The data in the bit timing registers are the configuration input of the CAN protocol controller. The Baud Rate Prescaler (configured by BRP) defines the length of the time quantum, the basic time unit of the bit time; the Bit Timing Logic (configured by TSeg1, TSeg2, and SJW) defines the number of time quanta in the bit time.

The processing of the bit time, the calculation of the position of the Sample Point, and occasional synchronizations are controlled by the BTL state machine, which is evaluated once each time quantum. The rest of the CAN protocol controller, the Bit Stream Processor (BSP) state machine is evaluated once each bit time, at the Sample Point.

The Shift Register serializes the messages to be sent and parallelizes received messages. Its loading and shifting is controlled by the BSP.

The BSP translates messages into frames and vice versa. It generates and discards the enclosing fixed format bits, inserts and extracts stuff bits, calculates and checks the CRC code, performs the error management, and decides which type of synchronization is to be used. It is evaluated at the Sample Point and processes the sampled bus input bit. The time after the Sample point that is needed to calculate the next bit to be sent (for example, data bit, CRC bit, stuff bit, error flag, or idle) is called the Information Processing Time (IPT).

The IPT is application specific but may not be longer than 2 t_q; the C-CAN's IPT is 0 t_q. Its length is the lower limit of the programmed length of Phase_Seg2. In case of a synchronization, Phase_Seg2 may be shortened to a value less than IPT, which does not affect bus timing.

Calculation of the bit timing parameters

Usually, the calculation of the bit timing configuration starts with a desired bitrate or bit time. The resulting bit time (1/ bit rate) must be an integer multiple of the system clock period.

The bit time may consist of 4 to 25 time quanta, the length of the time quantum t_q is defined by the Baud Rate Prescaler with t_q = (Baud Rate Prescaler) / f_{sys}. Several combinations may lead to the desired bit time, allowing iterations of the following steps.

First part of the bit time to be defined is the Prop_Seg. Its length depends on the delay times measured in the system. A maximum bus length as well as a maximum node delay has to

be defined for expandible CAN bus systems. The resulting time for Prop_Seg is converted into time quanta (rounded up to the nearest integer multiple of t_q).

The Sync_Seg is 1 t_q long (fixed), leaving (bit time – Prop_Seg – 1) t_q for the two Phase Buffer Segments. If the number of remaining t_q is even, the Phase Buffer Segments have the same length, Phase_Seg2 = Phase_Seg1, else Phase_Seg2 = Phase_Seg1 + 1.

The minimum nominal length of Phase_Seg2 has to be regarded as well. Phase_Seg2 may not be shorter than the CAN controller's Information Processing Time, which is, depending on the actual implementation, in the range of [0...2] t_q.

The length of the Synchronization Jump Width is set to its maximum value, which is the minimum of 4 and Phase_Seg1.

The oscillator tolerance range necessary for the resulting configuration is calculated by the formulas given in *System clock tolerance range on page 457*.

If more than one configuration is possible, that configuration allowing the highest oscillator or PLL tolerance range should be chosen.

CAN nodes with different system clocks require different configurations to come to the same bitrate. The calculation of the propagation time in the CAN network, based on the nodes with the longest delay times, is done once for the whole network.

The CAN system's oscillator (or PLL when used) tolerance range is limited by that node with the lowest tolerance range.

The calculation may show that bus length or bitrate have to be decreased or that the oscillator frequencies' stability has to be increased in order to find a protocol compliant configuration of the CAN bit timing.

The resulting configuration is written into the Bit Timing Register:

(Phase_Seg2 - 1) & (Phase_Seg1 + Prop_Seg - 1) & (SynchronizationJumpWidth - 1) & (Prescaler - 1)

Example for bit timing at high baudrate

In this example, the CPU frequency (CAN module clock) is 10 MHz, **BRP** is 0, the bitrate is 1 Mbit/s.

t_q	100	ns	=	t_{CPU}
Delay of bus driver	50	ns		
Delay of receiver circuit	30	ns		
Delay of bus line (40m)	220	ns		
t_{Prop}	600	ns	=	6 x t_q
t_{SJW}	100	ns	=	1 x t_q
t_{PB1}	100	ns	=	1 x t_q
$t_{Seg1} = t_{Prop} + t_{PB1}$	700	ns	= 7 x t_q	
$t_{Seg2} = t_{PB2}$	200	ns	=	Information Processing Time + 1 x t_q = 2 x t_q
$t_{Sync-Seg}$	100	ns	=	1 x t_q
t_{BT}	1000	ns	=	$t_{Sync-Seg} + t_{Seg1} + t_{Seg2}$ = 10 x t_q

Tolerance for CAN clock	0.39	%	=	$\frac{\min(t_{PB1},t_{PB2})}{2 \cdot (13 \cdot t_{BT} - t_{PB2})} = \frac{0.1\mu s}{2 \cdot (13 \cdot 1\mu s - 0.2\mu s)}$
δ_{PLL} (13 x t_{BT} = 13 x 10 x t_q = 130 t_{CPU})	5	ns	=	Data from PLL jitter characteristics
Tolerance for oscillator (no PLL effect)	0.35	%	=	$\frac{\min(t_{PB1},t_{PB2}) - 2 \cdot \delta_{PLL}}{2 \cdot (13 \cdot t_{BT} - t_{PB2})}$

In this example, the concatenated bit time parameters are $(2\text{-}1)_3$ & $(7\text{-}1)_4$ & $(1\text{-}1)_2$ & $(1\text{-}1)_6$, the Bit Timing Register CANxBTR is programmed to = 0x1600h.

Example for bit timing at low baudrate

In this example, the frequency of CAN module clock is 2 MHz, **BRP** is 1, the bitrate is 100 Kbit/s.

t_q	1	µs	=	2 x t_{CPU}
Delay of bus driver	200	ns		
Delay of receiver circuit	80	ns		
Delay of bus line (40m)	220	ns		
t_{Prop}	1	µs	=	1 x t_q
t_{SJW}	4	µs	=	4 x t_q
t_{PB1}	4	µs	=	4 x t_q
$t_{Seg1} = t_{Prop} + t_{PB1}$	5	µs	=	5 x t_q
$t_{Seg2} = t_{PB2}$	4	µs	=	Information Processing Time + 3 x t_q = 4 x t_q
$t_{Sync\text{-}Seg}$	1	µs	=	1 x t_q
t_{BT}	10	µs	=	$t_{Sync\text{-}Seg} + t_{Seg1} + t_{Seg2}$ = 10 x t_q
Tolerance for CAN clock	1.58	%	=	$\frac{\min(t_{PB1},t_{PB2})}{\cdot (13 \cdot t_{BT} - t_{PB2})} = \frac{4\mu s}{2 \cdot (13 \cdot 10\mu s - 4\mu s}$
δ_{PLL} (13 x t_{BT} = 13 x 10 x t_q = 260 t_{CPU})	10	ns	=	Data from PLL jitter characteristics
Tolerance for oscillator (no PLL effect)	1.57	%	=	$\frac{\min(t_{PB1},t_{PB2}) - 2 \cdot \delta_{PLL}}{2 \cdot (13 \cdot t_{BT} - t_{PB2})}$

In this example, the concatenated bit time parameters are $(4\text{-}1)_3$ & $(5\text{-}1)_4$ & $(4\text{-}1)_2$ & $(2\text{-}1)_6$, the Bit Timing Register CANxBTR is programmed to = 0x34C1h.

22 Real time clock

The Real Time Clock is an independent timer, which clock is directly derived from the oscillator clock (either the main on-chip oscillator or the 32 kHz on-chip oscillator), so that it can be maintained running even in Idle or Power Down mode (if enabled to) or again in Stand-by mode. Registers access is implemented onto the XBUS. This module is designed for the following purposes:

- Generate the current time and date for the system.
- Provide cyclic time based interrupt on Port2 external interrupts every 'RTC basic clock tick' and after *n* 'RTC basic clock ticks' (*n* is programmable) if enabled.
- Long term measurements (58-bit timer).
- Exit the ST10F276 from Power Down mode (if PWDCFG of SYSCON set) after a programmed delay.

The real time clock is based on two main blocks of counters. The first block is a prescaler which generates a basic reference clock (for example a 1 second period). This basic reference clock is coming out of a 20-bit DIVIDER (4-bit MSB RTCDH counter and 16-bit LSB RTCDL counter). This 20-bit counter is driven by an input clock derived from the on-chip oscillator clock (XTAL1 input), pre-divided by a 1/64 fixed counter (see *Figure 189*). This 20-bit counter is loaded at each basic reference clock period with the value of the 20-bit PRESCALER register (4-bit MSB RTCPH register and 16-bit LSB RTCPL register). The value of the 20-bit RTCP register determines the period of the basic reference clock.

A timed interrupt request (RTCSI) may be sent on each basic reference clock period. The second block of the RTC is a 32-bit counter (16-bit RTCH and 16-bit RTCL). This counter may be initialized with the current system time. RTCH/RTCL counter is driven with the basic reference clock signal. In order to provide an alarm function the contents of RTCH/RTCL counter is compared with a 32-bit alarm register (16-bit RTCAH register and 16-bit RTCAL register). The alarm register may be loaded with a reference date. An alarm interrupt request (RTCAI), may be generated when the value of RTCH/RTCL counter matches the reference date of RTCAH/RTCAL register.

The timed RTCSI and the alarm RTCAI interrupt requests can trigger a fast external interrupt via EXISEL register of Port2 and wakes the ST10 up when running in Power Down mode. Using the RTCOFF bit of RTCCON register, the user may switch off the clock oscillator when entering the Power Down mode.

Figure 187. SFRs associated with the RTC

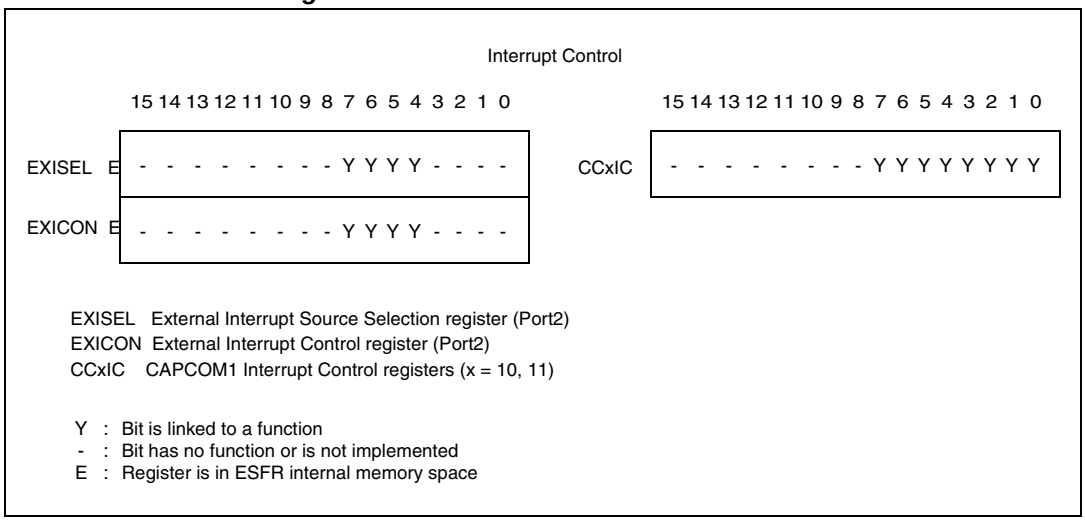


Figure 188. XBUS registers associated with the RTC

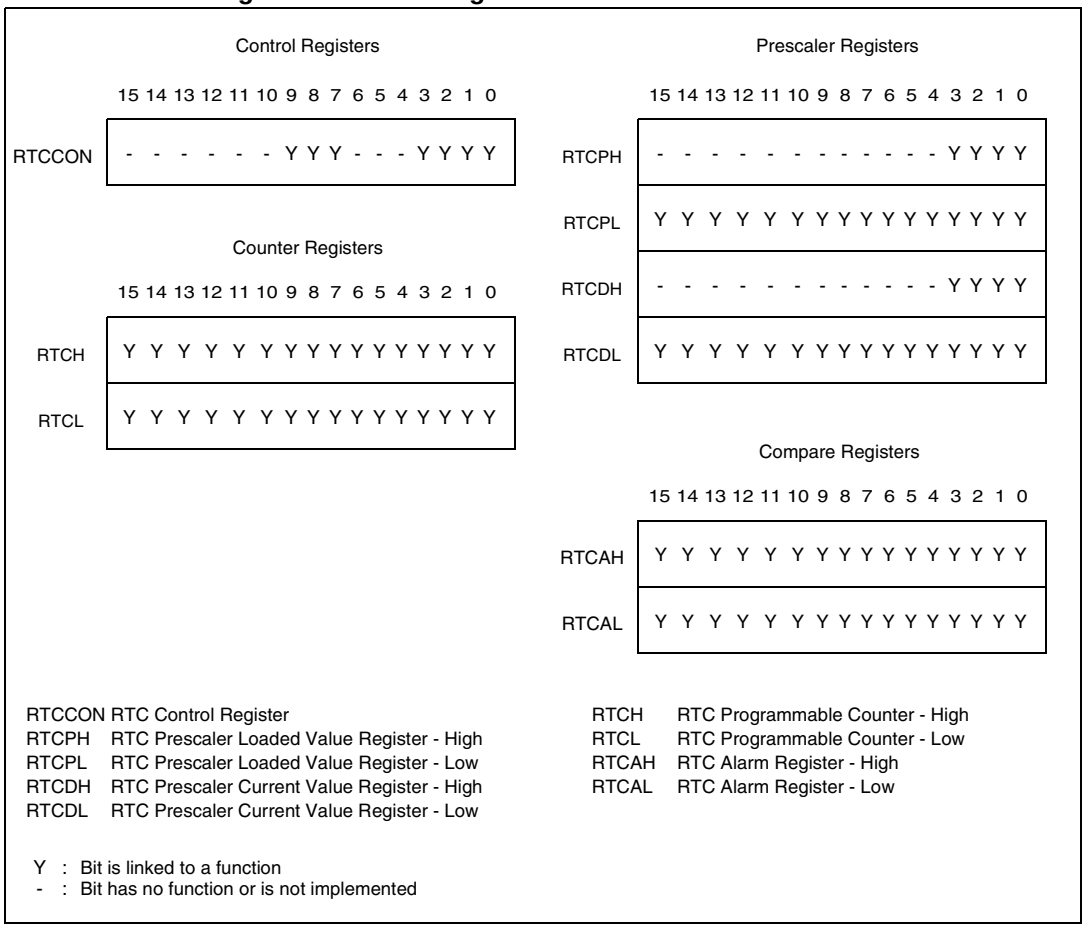
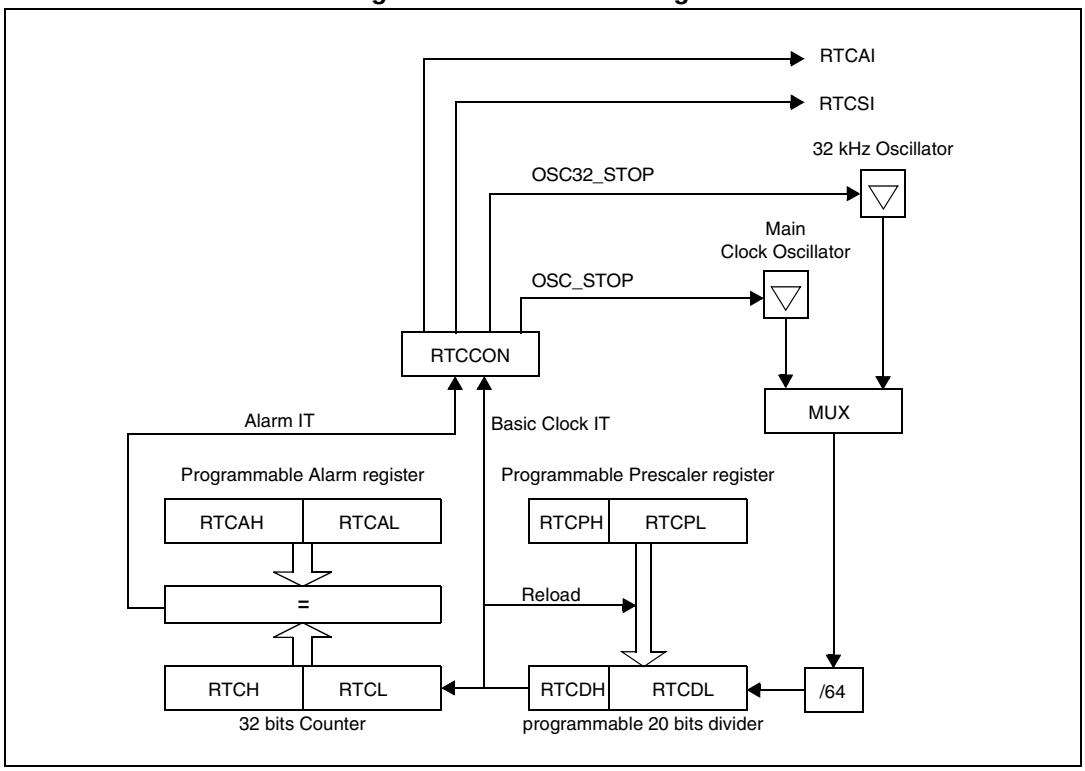


Figure 189. RTC block diagram



22.1 RTC registers

22.1.1 RTCCON: RTC control register

The functions of the RTC are controlled by the RTCCON control register. If the RTOFF bit is set, the RTC dividers and counters clock is disabled and registers can be written, when the ST10 chip enters Power Down mode the clock oscillator will be switch off. The RTC has two interrupt sources, one is triggered every basic clock period, the other one is the alarm.

RTCCON includes an interrupt request flag and an interrupt enable bit for each of them. This register is read and written via the XBUS.

RTCCON (ED00h) XBUS Reset Value: 0x00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	OFF32	**OSC**	RTCOFF	-	-	-	RTCAEN	**RTCAIR**	RTCSEN	**RTCSIR**
						RW	R	RW				RW	RW	RW	RW

Reset Value: 0000 000x 0000 0000b

Bit	Function
RTCSIR	**RTC Second Interrupt Request flag** '0': the bit was reset less than one basic clock tick ago. '1': the interrupt was triggered.
RTCSEN	**RTC Second interrupt Enable** '0': RTCSI is disabled. '1': RTCSI is enabled, it is generated every basic clock tick.
RTCAIR	**RTC Alarm Interrupt Request flag** (when the alarm is triggered) '0': the bit was reset less than a *n* basic clock tick ago. '1': the interrupt was triggered.
RTCAEN	**RTC Alarm Interrupt Enable** '0': RTCAI is disabled. '1': RTCAI is enabled, it is generated when the counters reach the alarm value.
RTCOFF	**RTC Switch Off bit** '0': clock oscillator and RTC are kept on running even if ST10 is in Power Down mode. '1': clock oscillator is switched off if ST10 enters Power Down mode. Besides, setting this bit RTC dividers and counters are stopped, and registers can be written.
OSC	**Oscillator Selection Flag** '0': the clock oscillator used by the RTC is the Main oscillator. '1': the clock oscillator used by the RTC is the low power 32 kHz oscillator.
OFF32	**32 kHz Oscillator Switch Off bit** '0': The 32 kHz oscillator is enabled. '1': The 32 kHz oscillator is disabled.

Note: All the bits of RTCCON are active high.

The two RTC Interrupt request lines are internally connected to Port2 input lines in order to trigger an external interrupt that wakes the chip up if in Power Down mode.

All the RTC registers are not bit addressable.

To clear the RTC Interrupt Request flags (bit 0 and bit 2 of the RTCCON register) it is necessary to write a '1' to the corresponding bit of the RTCCON register.

22.1.2 RTCPH & RTCPL: RTC prescaler registers

The 20-bit programmable prescaler divider is loaded with two registers.

The 4 most significant bits are stored into RTCPH and the 16 less significant bits are stored in RTCPL. In order to keep the system clock, those registers are not reset.

They are write protected by bit RTOFF of RTCCON register, write operation is allowed if RTOFF is set.

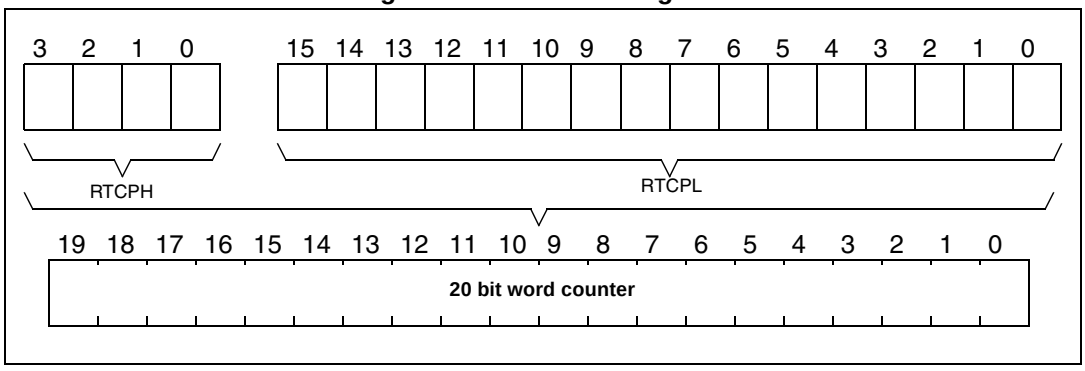
RTCPL (ED06h) XBUS Reset Value: xxxxh

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

RTCPH (ED08h) XBUS Reset Value: - - - xh

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

Figure 190. Prescaler register



The value stored into RTCPH, RTCPL is called RTCP (coded on 20-bit). The dividing ratio of the Prescaler divider is:

ratio = 64 x (RTCP)

The minimum value which can be set in RTCPL is 0002h.

22.1.3 RTCDH & RTCDL: RTC divider counters

Every basic reference clock the Divider counters are reloaded with the value stored RTCPH and RTCPL registers. To get an accurate time measurement it is possible to read the value of the Divider, reading the RTCDH, RTCDL. Those counters are read only. After any bit changed in the programmable Prescaler register, the new value is loaded in the Divider.

RTCDL (ED0Ah) XBUS Reset Value: xxxxh

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

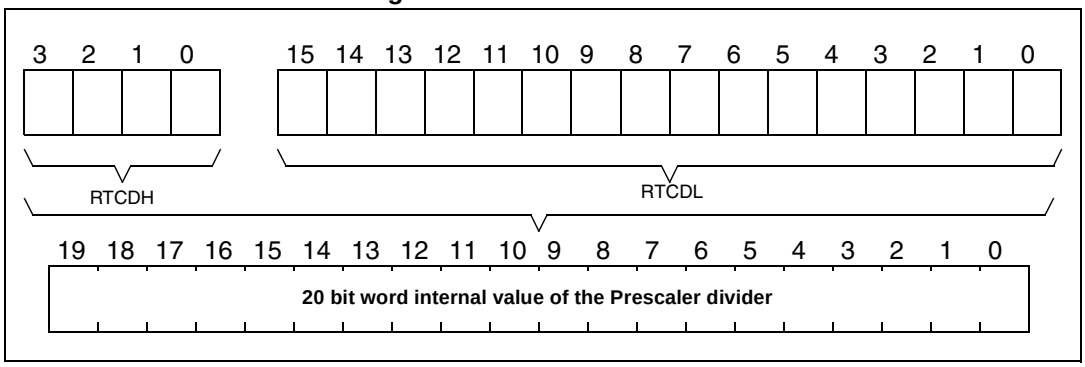
RTCDH (ED0Ch) XBUS Reset Value: - - - xh

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

Note: These registers are not reset, and are read only.

The divider works as a decrementor: when the internal value reaches 0001h, the second interrupt is generated. Then, when next decrement occurs (which would virtually put in the divider register the value 0000h), the 20-bit word stored into RTCPH, RTCPL registers is loaded in the divider. As already stated, the minimum value which can be programmed in RTCPL is 0002h: if 0001h were set, just one second interrupt would be generated, since the divider would stay fixed at the value 0001h forever (successive second interrupt cannot occur).

Figure 191. Divider counters



Bit 15 to bit 4 of RTCPH and RTCDH are not used. When reading, the return value of those bit will be zeros.

22.1.4 RTCH & RTCL: RTC programmable counter registers

The RTC has 2 x 16-bit programmable counters which count rate is based on the basic time reference (for example 1 second). As the clock oscillator may be kept working, even in Power Down mode, the RTC counters may be used as a clock for real time system date (either the main or the 32 kHz oscillator). In addition RTC counters and registers are not modified at any system reset. The only way to force their value is to write them via the XBUS.

These counters are write protected as well. The bit RTOFF of the RTCCON register must be set (RTC dividers and counters are stopped) to enable a write operation on RTCH or RTCL.

A write operation on RTCH or RTCL register loads directly the corresponding counter. When reading, the current value in the counter (system date) is returned.

The counters keeps on running while the clock oscillator is working.

RTCL (ED0Eh) XBUS Reset Value: xxxxh

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

RTCH (ED10h) XBUS Reset Value: xxxxh

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

Note: These registers are not reset.

22.1.5 RTCAH & RTCAL: RTC alarm registers

When the programmable counters reach the 32-bit value stored into RTCAH & RTCAL registers, an alarm is triggered and the interrupt request RTAIR is generated. These registers are not protected.

RTCAL (ED12h) XBUS Reset Value: xxxxh

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

RTCAH (ED14h) XBUS Reset Value: xxxxh

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

Note: These registers are not reset.

22.2 Programming the RTC

RTC interrupt request signals are connected to Port2, pin 10 (RTCSI) and pin 11 (RTCAI). An alternate function of Port2 is to generate fast interrupts firq[7:0]. To trigger firq[2] and firq[3] the following configuration has to be set.

EXICON ESFR controls the external interrupt edge selection, RTC interrupt requests are rising edge active.

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

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

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

Note: 1. EXI2ES and EXI3ES must be configured as '01b' because RTC interrupt request lines are rising edge active.
2. Alarm interrupt request line (RTCAI) is linked with EXI3ES.
3. Timed interrupt request line (RTCSI) is linked with EXI2ES.

EXISEL ESFR enables the Port2 alternate sources. RTC interrupts are alternate sources 2 and 3.

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

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

Bit	Function
EXIxSS	**External Interrupt x Source Selection** (x = 7...0) ‘00’: Input from associated Port2 pin. ‘01’: Input from “alternate source”. [1] ‘10’: Input from Port2 pin ORed with “alternate source”. [1] ‘11’: Input from Port2 pin ANDed with “alternate source”.

Note: Recommended configurations.

Interrupt control register are common with CAPCOM1 Unit: CC10IC (RTCSI) and CC11IC (RTCAI).

CC10IC / CC11IC SFR Reset Value: - - 00h

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

CC10IC: FF8Ch/C6h

CC11IC: FF8Eh/C7h

Source of interrupt	Request flag	Enable flag	Interrupt vector	Vector location	Trap number
External interrupt 2	CC10IR	CC10IE	CC10INT	00’0068h	1Ah / 26
External interrupt 3	CC11IR	CC11IE	CC11INT	00’006Ch	1Bh / 27

23 System reset

System reset initializes the device in a predefined state. There are many ways to activate a reset state. The system start-up configuration is different for each case as shown in *Table 63*. The reset history is flagged inside WDTCON register (see also *Section 14: Watchdog timer on page 297* for additional details).

Table 63. Reset event definition

Reset source	Flag	RPD status	Conditions
Power-On reset	PONR	Low	Power-On
Asynchronous Hardware reset	LHWR	Low	$t_{\overline{RSTIN}}$ > (1)
Synchronous Long Hardware reset		High	$t_{\overline{RSTIN}}$ > (1032 + 12) TCL + max(4 TCL, 500ns)
Synchronous Short Hardware reset	SHWR	High	$t_{\overline{RSTIN}}$ > max(4 TCL, 500ns) $t_{\overline{RSTIN}}$ ≤ (1032 + 12) TCL + max(4 TCL, 500ns)
Watchdog Timer reset	WDTR	(2)	WDT overflow
Software reset	SWR	note (2)	SRST instruction execution

1. $\overline{RSTIN}$ pulse should be longer than 500ns (Filter) and than settling time for configuration of Port0. See next Section 24.1 for more details on minimum reset pulse duration.
2. The RPD status has no influence unless Bidirectional Reset is activated (bit BDRSTEN in SYSCON): RPD low inhibits the Bidirectional reset on SW and WDT reset events, that is RSTIN is not activated (refer to *Section 23.4: Software reset*, *Section 23.5: Watchdog timer reset* and *Section 23.6: Bidirectional reset*).

23.1 Input filter

On $\overline{RSTIN}$ input pin an on-chip RC filter is implemented. It is sized to filter all the spikes shorter than 50ns. On the other side, a valid pulse should be longer than 500ns to grant that ST10 recognizes a reset command. In between 50ns and 500ns a pulse can either be filtered or recognized as valid, depending on the operating conditions and process variations.

For this reason all minimum durations mentioned in this section for the different kind of reset events should be carefully evaluated taking into account of the above requirements.

In particular, for Short Hardware Reset, where only 4 TCL is specified as minimum input reset pulse duration, the operating frequency is a key factor. Examples:

- for a CPU clock of 64 MHz, 4 TCL is 31.25ns, so it would be filtered: in this case the minimum becomes the one imposed by the filter (that is 500ns).
- for a CPU clock of 4 MHz, 4 TCL is 500ns: in this case the minimum from the formula is coherent with the limit imposed by the filter.

23.2 Asynchronous reset

An asynchronous reset is triggered when $\overline{RSTIN}$ pin is pulled low while RPD pin is at low level. Then the ST10F276 is immediately (after the input filter delay) forced in reset default state. It pulls low $\overline{RSTOUT}$ pin, it cancels pending internal hold states if any, it aborts all

internal/external bus cycles, it switches buses (data, address and control signals) and I/O pin drivers to high-impedance, it pulls high Port0 pins.

Note: If an asynchronous reset occurs during a read or write phase in internal memories, the content of the memory itself could be corrupted: to avoid this, synchronous reset usage is strongly recommended.

Power-on reset

The asynchronous reset must be used during the Power-On of the device. Depending on crystal or resonator frequency, the on-chip oscillator needs about 1ms to 10ms to stabilize (Refer to ST10F276x datasheet - Electrical Characteristics Section), with an already stable V_{DD}. The logic of the ST10F276 does not need a stabilized clock signal to detect an asynchronous reset, so it is suitable for Power-On conditions. To ensure a proper reset sequence, the $\overline{\text{RSTIN}}$ pin and the RPD pin must be held at low level until the device clock signal is stabilized and the system configuration value on Port0 is settled.

At Power-On it is important to respect some additional constraints introduced by the start-up phase of the different embedded modules.

In particular the on-chip voltage regulator needs at least 1ms to stabilize the internal 1.8V for the core logic: this time is computed from when the external reference (V_{DD}) becomes stable (inside specification range, that is at least 4.5V). This is a constraint for the application hardware (external voltage regulator): the $\overline{\text{RSTIN}}$ pin assertion should be extended to guarantee the voltage regulator stabilization.

A second constraint is imposed by the embedded Flash. When booting from internal memory, starting from $\overline{\text{RSTIN}}$ releasing, it needs a maximum of 1ms for its initialization: before that, the internal reset (RST signal) is not released, so the CPU does not start code execution in internal memory.

Note: This is not true if external memory is used (pin $\overline{EA}$ held low during reset phase). In this case, once $\overline{RSTIN}$ pin is released, and after few CPU clock (Filter delay plus 3...8 TCL), the internal reset signal RST is released as well, so the code execution can start immediately after. Obviously, an eventual access to the data in internal Flash is forbidden before its initialization phase is completed: an eventual access during starting phase will return FFFFh (just at the beginning), while later 009Bh (an illegal opcode trap can be generated).

At Power-On, the $\overline{\text{RSTIN}}$ pin should be tied low for a minimum time that includes also the start-up time of the main oscillator (t_{STUP} = 1ms for resonator, 10ms for crystal) and PLL synchronization time (t_{PSUP} = 200µs): this means that if the internal Flash is used, the $\overline{\text{RSTIN}}$ pin could be released before the main oscillator and PLL are stable to recover some time in the start-up phase (Flash initialization only needs stable V_{18}, but does not need stable system clock since an internal dedicated oscillator is used).

Caution: It is recommended to provide the external hardware with a current limitation circuitry. This is necessary to avoid permanent damages of the device during the Power-On transient, when the capacitance on V_{18} pin is charged. For the on-chip voltage regulator functionality 10nF are sufficient: anyway, a maximum of 100nF on V_{18} pin should not generate problems of over-current (higher value is allowed if current is limited by the external hardware). External current limitation is anyway recommended also to avoid risks of damage in case of temporary short between V_{18} and ground: the internal 1.8V drivers are sized to drive currents of several tens of Ampere, so the current should be limited by the external hardware. The limit of current is imposed by power dissipation considerations (Refer to ST10F276x datasheet - Electrical Characteristics Section).

Figure 192 and *Figure 193* show Asynchronous Power-On timing diagrams, respectively with boot from internal or external memory, highlighting the reset phase extension introduced by the embedded Flash module when selected.

Note: Never power the device without keeping $\overline{RSTIN}$ pin grounded: the device could enter in unpredictable states, risking also permanent damages.

Figure 192. Asynchronous power-on RESET ($\overline{EA}$ = 1)

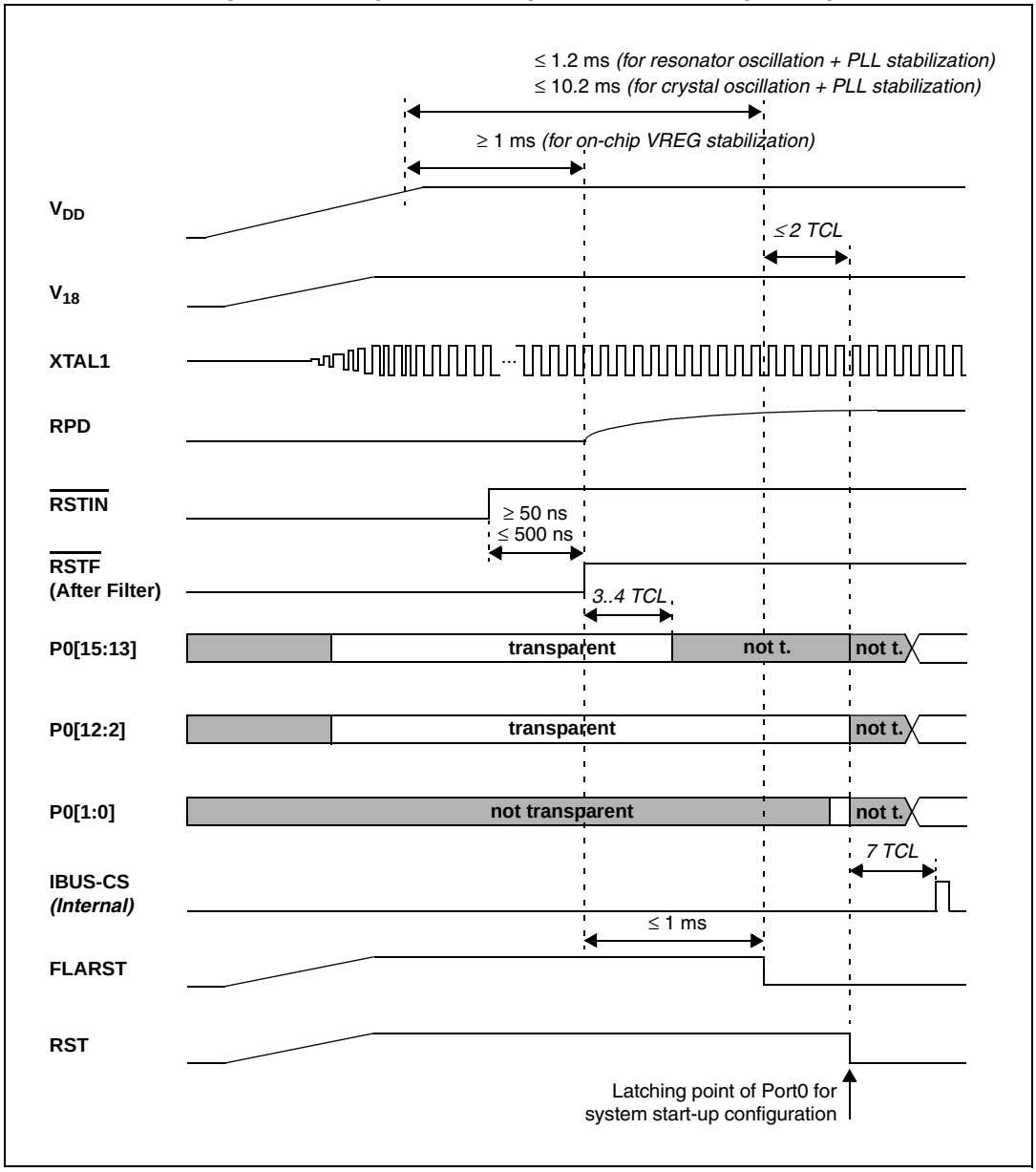
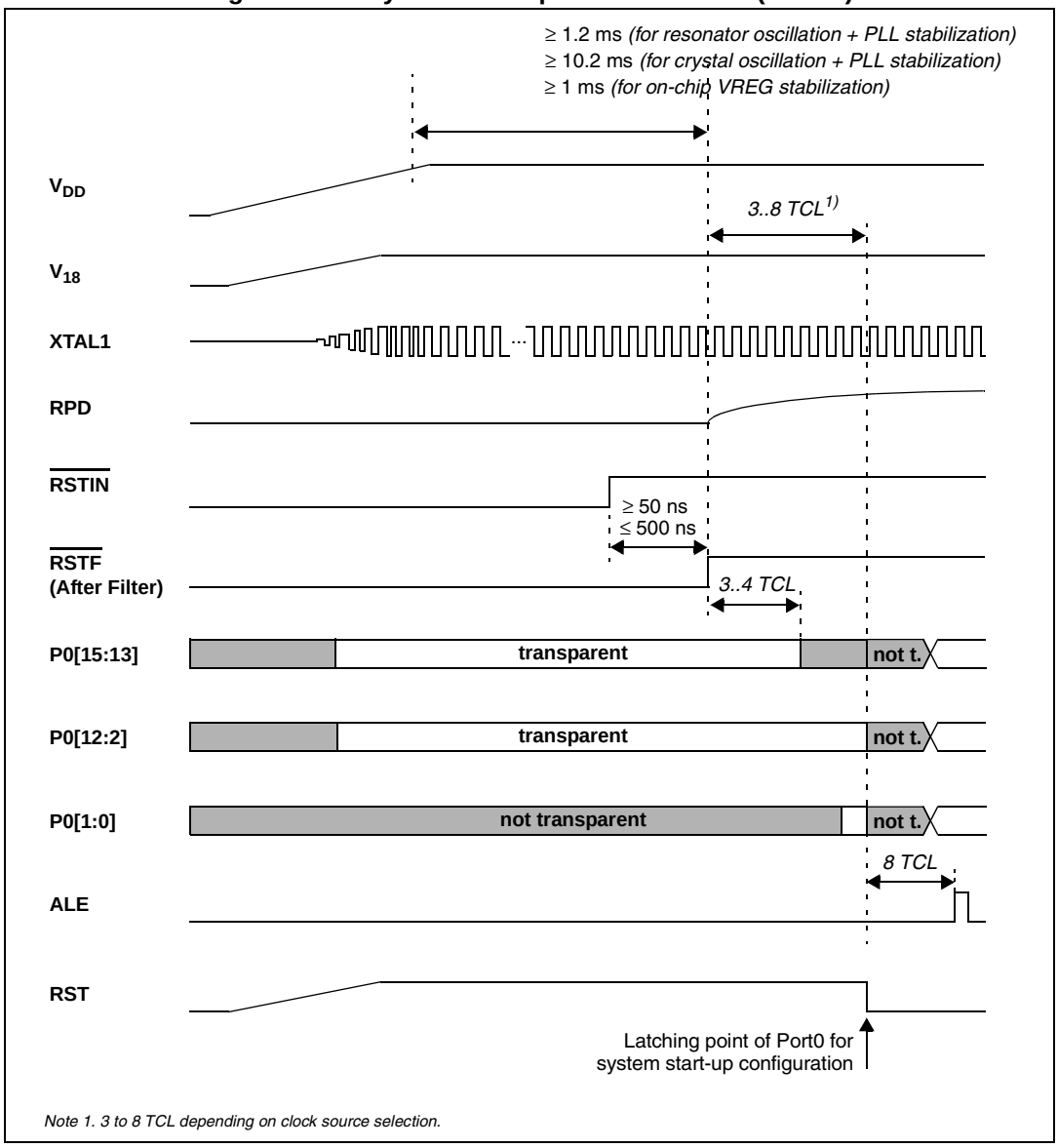


Figure 193. Asynchronous power-on RESET ($\overline{EA}$ = 0)

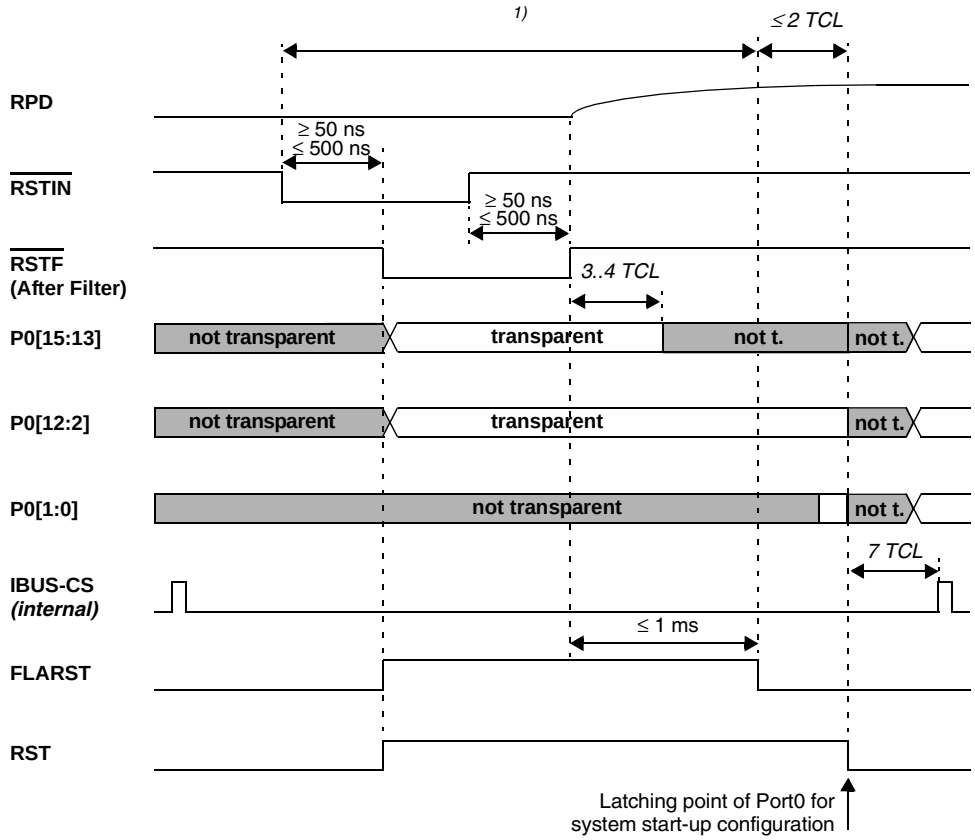


Note 1. 3 to 8 TCL depending on clock source selection.

Hardware reset

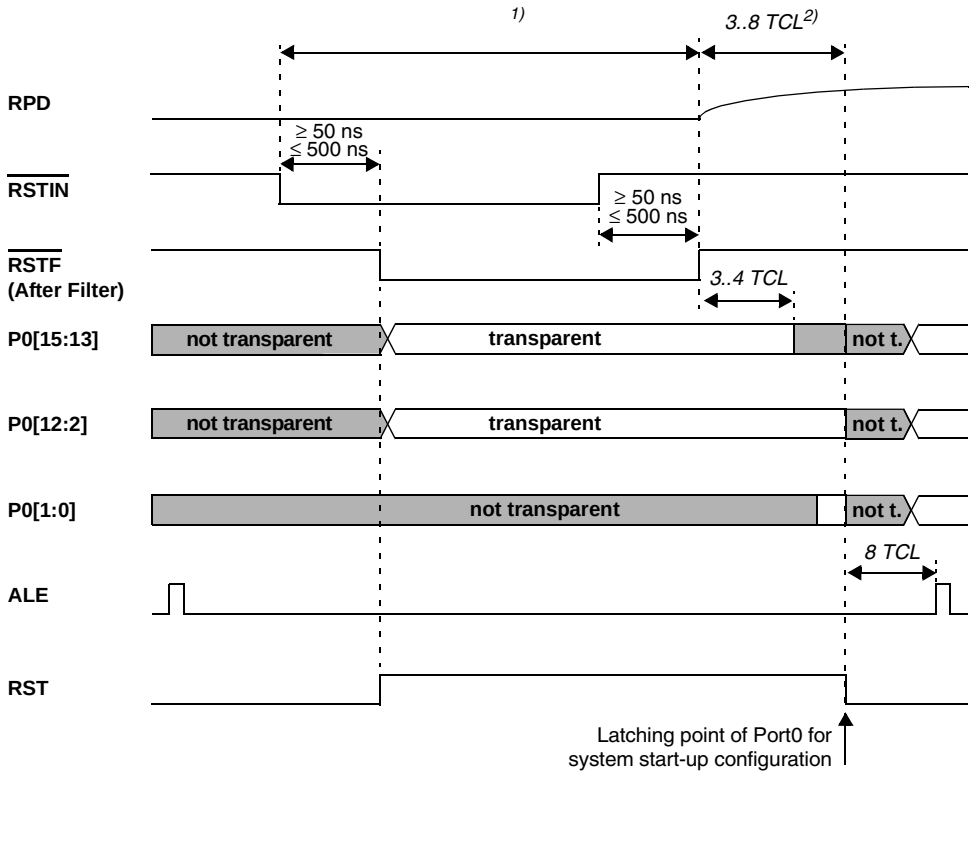
The asynchronous reset must be used to recover from catastrophic situations of the application. It may be triggered by the hardware of the application. Internal hardware logic and application circuitry are described in *Section 23: System reset on page 472* and in *Figure 205 on page 491*, *Figure 206 on page 492* and *Figure 3 on page 30*. It occurs when $\overline{RSTIN}$ is low and RPD is detected (or becomes) low as well.

Figure 194. Asynchronous hardware RESET ($\overline{EA}$ = 1)



Note 1. Longer than Port0 settling time + PLL synchronization (if needed, that is P0(15:13) changed)
Longer than 500ns to take into account of Input Filter on $\overline{RSTIN}$ pin

Figure 195. Asynchronous hardware RESET ($\overline{EA}$ = 0)



Note 1. Longer than Port0 settling time + PLL synchronization (if needed, that is P0(15:13) changed)
Longer than 500ns to take into account of Input Filter on $\overline{RSTIN}$ pin

Note 2. 3 to 8 TCL depending on clock source selection.

Exit from asynchronous reset state

When the $\overline{RSTIN}$ pin is pulled high, the device restarts: as already mentioned, if internal Flash is used, the restarting occurs after the embedded Flash initialization routine is completed. The system configuration is latched from Port0: ALE, $\overline{RD}$ and $\overline{WR}$/$\overline{WRL}$ pins are driven to their inactive level. The ST10F276 starts program execution from memory location 00'0000h in code segment 0. This starting location will typically point to the general initialization routine. Timing of asynchronous Hardware Reset sequence are summarized in *Figure 194* and *Figure 195*.

23.3 Synchronous reset (warm reset)

A synchronous reset is triggered when $\overline{RSTIN}$ pin is pulled low while RPD pin is at high level. In order to properly activate the internal reset logic of the device, the $\overline{RSTIN}$ pin must be held low, at least, during 4 TCL (2 periods of CPU clock): refer also to *Section 23.1: Input filter* for details on minimum reset pulse duration. The I/O pins are set to high impedance and $\overline{RSTOUT}$ pin is driven low. After $\overline{RSTIN}$ level is detected, a short duration of a maximum of 12 TCL (6 periods of CPU clock) elapses, during which pending internal hold states are cancelled and the current internal access cycle if any is completed. External bus cycle is

aborted. The internal pull-down of RSTIN pin is activated if bit BDRSTEN of SYSCON register was previously set by software. Note that this bit is always cleared on Power-On or after a reset sequence.

Short and long synchronous reset

Once the first maximum 16 TCL are elapsed (4+12TCL), the internal reset sequence starts. It is 1024 TCL cycles long: at the end of it, and after other 8TCL the level of RSTIN is sampled (after the filter, see RSTF in the drawings): if it is already at high level, only Short Reset is flagged (Refer to *Section 14: Watchdog timer on page 297* for details on reset flags); if it is recognized still low, the Long reset is flagged as well. The major difference between Long and Short reset is that during the Long reset, also P0(15:13) become transparent, so it is possible to change the clock options.

Note: In case of a short pulse on RSTIN pin, and when Bidirectional reset is enabled, the RSTIN pin is held low by the internal circuitry. At the end of the 1024 TCL cycles, the RTSIN pin is released, but due to the presence of the input analog filter the internal input reset signal (RSTF in the drawings) is released later (from 50 to 500ns). This delay is in parallel with the additional 8 TCL, at the end of which the internal input reset line (RSTF) is sampled, to decide if the reset event is Short or Long. In particular:

- If 8 TCL > 500ns (f_{CPU} < 8 MHz), the reset event is always recognized as Short
- If 8 TCL < 500ns (f_{CPU} > 8 MHz), the reset event could be recognized either as Short or Long, depending on the real filter delay (between 50 and 500ns) and the CPU frequency (RSTF sampled High means Short reset, RSTF sampled Low means Long reset). Note that in case a Long Reset is recognized, once the 8 TCL are elapsed, the P0(15:13) pins becomes transparent, so the system clock can be re-configured. The port returns not transparent 3-4TCL after the internal RSTF signal becomes high.

The same behavior just described, occurs also when unidirectional reset is selected and RSTIN pin is held low till the end of the internal sequence (exactly 1024TCL + max 16 TCL) and released exactly at that time.

Note: When running with CPU frequency lower than 40 MHz, the minimum valid reset pulse to be recognized by the CPU (4 TCL) could be longer than the minimum analog filter delay (50ns); so it might happen that a short reset pulse is not filtered by the analog input filter, but on the other hand it is not long enough to trigger a CPU reset (shorter than 4 TCL): this would generate a Flash reset but not a system reset. In this condition, the Flash always answers with FFFFh, which leads to an illegal opcode and consequently a trap event is generated.

Exit from synchronous reset state

The reset sequence is extended until RSTIN level becomes high. Besides, it is internally prolonged by the Flash initialization when EA = 1 (internal memory selected). Then, the code execution restarts. The system configuration is latched from Port0, and ALE, RD and WR/WRL pins are driven to their inactive level. The ST10F276 starts program execution from memory location 00'0000h in code segment 0. This starting location will typically point to the general initialization routine. Timing of synchronous reset sequence are summarized in *Figure 196* and *Figure 197* where a Short Reset event is shown, with particular highlighting on the fact that it can degenerate into Long Reset: the two figures show the behavior when booting from internal or external memory respectively. *Figure 198* and *Figure 199* report the timing of a typical synchronous Long Reset, again when booting from internal or external memory.

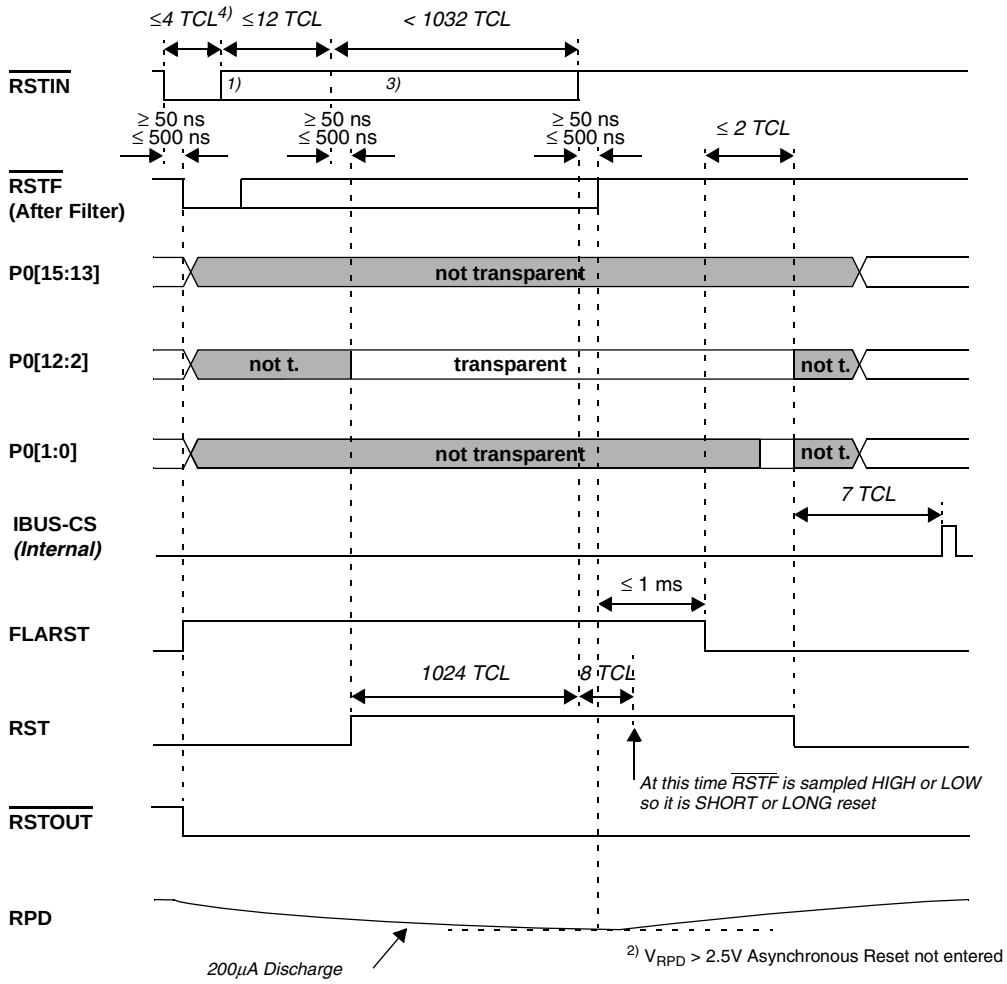
Synchronous reset and RPD pin

Whenever the $\overline{\text{RSTIN}}$ pin is pulled low (by external hardware or as a consequence of a Bidirectional reset), the RPD internal weak pull-down is activated. The external capacitance (if any) on RPD pin is slowly discharged through the internal weak pull-down. If the voltage level on RPD pin reaches the input low threshold (around 2.5V), the reset event becomes immediately asynchronous. In case of hardware reset (short or long) the situation goes immediately to the one illustrated in *Figure 194 on page 476*. There is no effect if RPD comes again above the input threshold: the asynchronous reset is completed coherently. To grant the normal completion of a synchronous reset, the value of the capacitance should be big enough to maintain the voltage on RPD pin sufficient high along the duration of the internal reset sequence.

For a Software or Watchdog reset events, an active synchronous reset is completed regardless of the RPD status.

It is important to highlight that the signal that makes RPD status transparent under reset is the internal $\overline{\text{RSTF}}$ (after the noise filter).

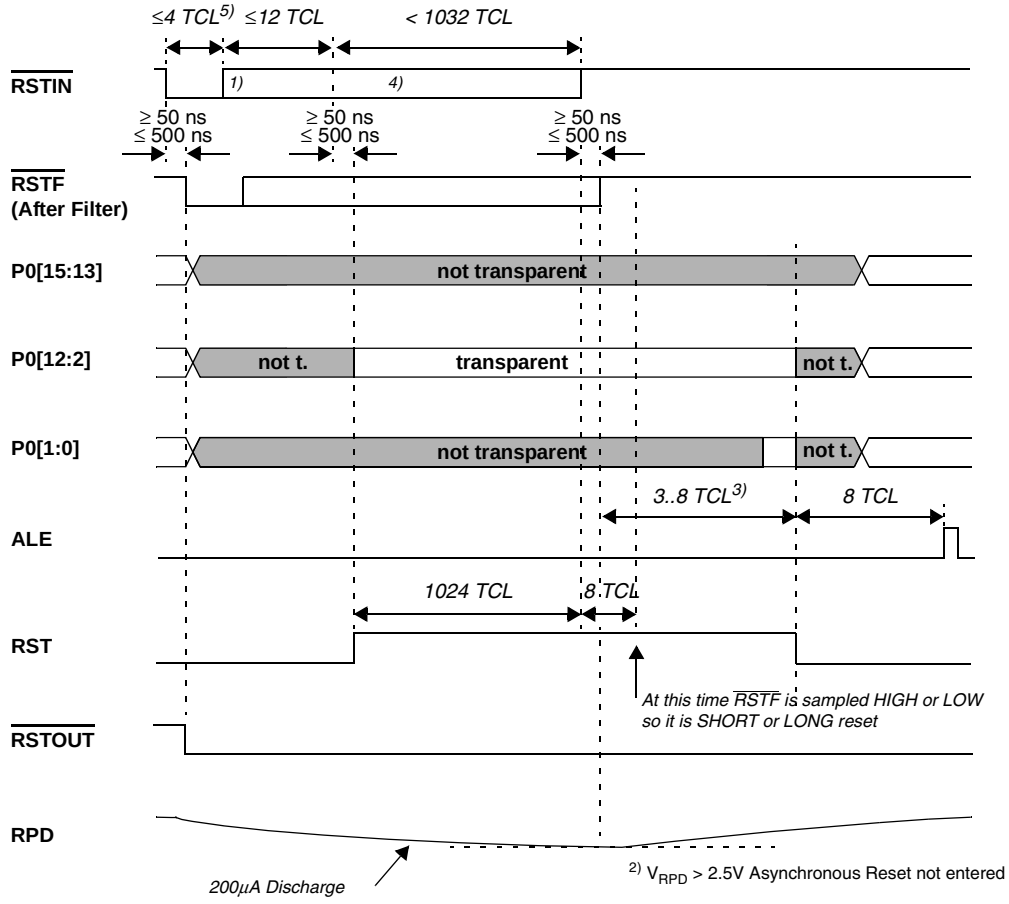
Figure 196. Synchronous short / long hardware RESET ($\overline{EA}$ = 1)



Notes:

1. $\overline{RSTIN}$ *assertion can be released there. Refer also to Section 21.1 for details on minimum pulse duration.*
2. *If during the reset condition ($\overline{RSTIN}$ low), RPD voltage drops below the threshold voltage (about 2.5V for 5V operation), the asynchronous reset is then immediately entered.*
3. $\overline{RSTIN}$ *pin is pulled low if bit BDRSTEN (bit 3 of SYSCON register) was previously set by software. Bit BDRSTEN is cleared after reset.*
4. *Minimum $\overline{RSTIN}$ low pulse duration should also be longer than 500ns to guarantee the pulse is not masked by the internal filter (refer to Section 21.1).*

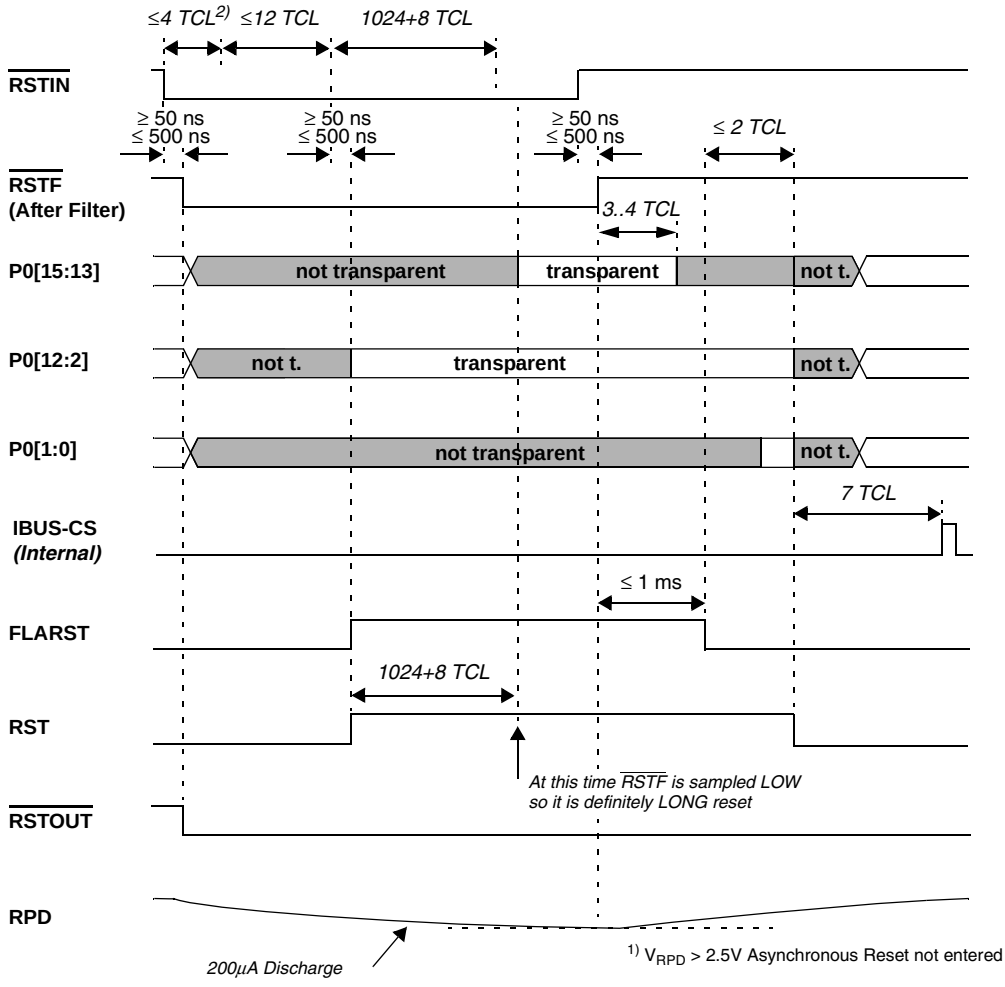
Figure 197. Synchronous short / long hardware RESET ($\overline{EA}$ = 0)



Notes:

1. $\overline{RSTIN}$ assertion can be released there. Refer also to Section 21.1 for details on minimum pulse duration.
2. If during the reset condition ($\overline{RSTIN}$ low), RPD voltage drops below the threshold voltage (about 2.5V for 5V operation), the asynchronous reset is then immediately entered.
3. 3 to 8 TCL depending on clock source selection.
4. $\overline{RSTIN}$ pin is pulled low if bit BDRSTEN (bit 3 of SYSCON register) was previously set by software. Bit BDRSTEN is cleared after reset.
5. Minimum $\overline{RSTIN}$ low pulse duration should also be longer than 500ns to guarantee the pulse is not masked by the internal filter (refer to Section 21.1).

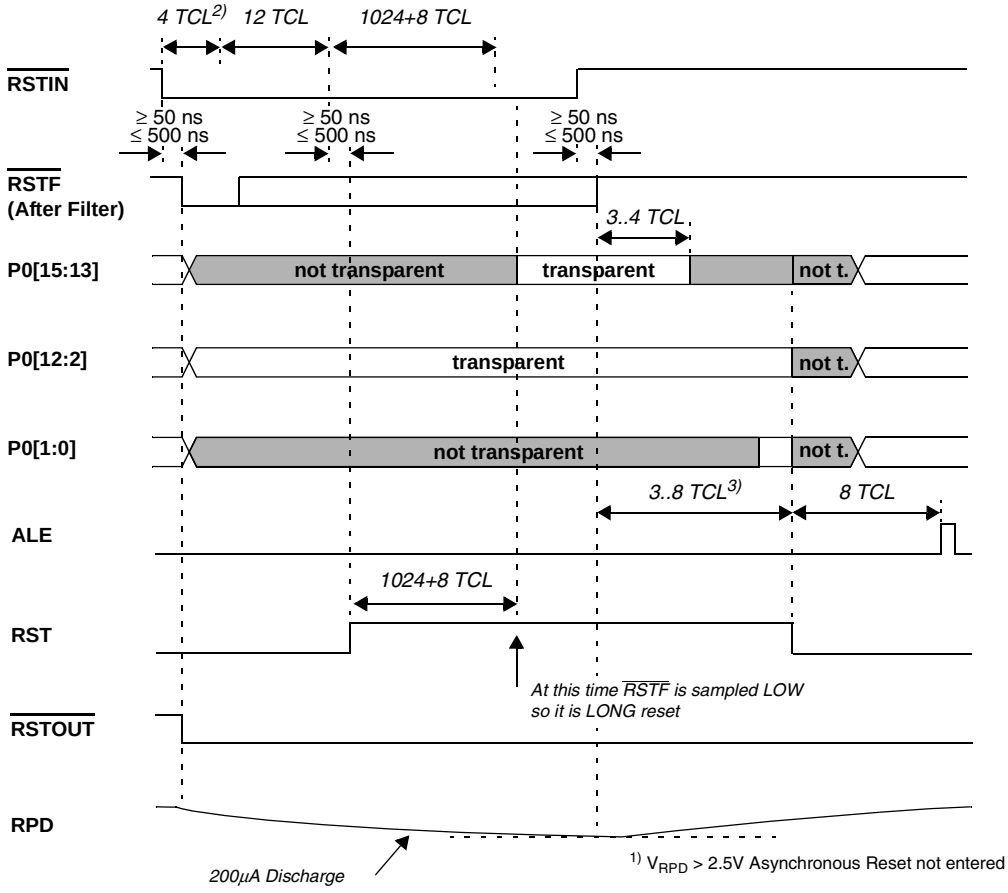
Figure 198. Synchronous long hardware RESET ($\overline{EA}$ = 1)



Notes:

1. *If during the reset condition ($\overline{RSTIN}$ low), RPD voltage drops below the threshold voltage (about 2.5V for 5V operation), the asynchronous reset is then immediately entered. Even if RPD returns above the threshold, the reset is defnitively taken as asynchronous.*
2. *Minimum $\overline{RSTIN}$ low pulse duration should also be longer than 500ns to guarantee the pulse is not masked by the internal filter (refer to Section 21.1).*

Figure 199. Synchronous long hardware RESET ($\overline{EA}$ = 0)



Notes:

1. If during the reset condition ($\overline{RSTIN}$ low), RPD voltage drops below the threshold voltage (about 2.5V for 5V operation), the asynchronous reset is then immediately entered.

2. Minimum $\overline{RSTIN}$ low pulse duration should also be longer than 500ns to guarantee the pulse is not masked by the internal filter (refer to Section 21.1).

3. 3 to 8 TCL depending on clock source selection.

23.4 Software reset

A software reset sequence can be triggered at any time by the protected SRST (software reset) instruction. This instruction can be deliberately executed within a program, for example, to leave bootstrap loader mode, or on a hardware trap that reveals system failure.

On execution of the SRST instruction, the internal reset sequence is started. The microcontroller behavior is the same as for a synchronous short reset, except that only bits P0.12...P0.8 are latched at the end of the reset sequence, while previously latched, bits P0.7...P0.2 are cleared (that is written at ‘1’).

A Software reset is always taken as synchronous: there is no influence on Software Reset behavior with RPD status. In case Bidirectional Reset is selected, a Software Reset event pulls RSTIN pin low: this occurs only if RPD is high; if RPD is low, RSTIN pin is not pulled low even though Bidirectional Reset is selected.

Refer to *Figure 200* and *Figure 201* for unidirectional SW reset timing, and to *Figure 202*, *Figure 203* and *Figure 204* for bidirectional reset timing.

23.5 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.

Unlike hardware and software resets, the watchdog reset completes a running external bus cycle if this bus cycle either does not use READY, or if READY is sampled active (low) after the programmed wait states.

When READY is sampled inactive (high) after the programmed wait states the running external bus cycle is aborted. Then the internal reset sequence is started.

Bit P0.12...P0.8 are latched at the end of the reset sequence and bit P0.7...P0.2 are cleared (that is written at ‘1’).

A Watchdog reset is always taken as synchronous: there is no influence on Watchdog Reset behavior with RPD status. In case Bidirectional Reset is selected, a Watchdog Reset event pulls RSTIN pin low: this occurs only if RPD is high; if RPD is low, RSTIN pin is not pulled low even though Bidirectional Reset is selected.

Refer to *Figure 200* and *Figure 201* for unidirectional SW reset timing, and to *Figure 202*, *Figure 203* and *Figure 204* for bidirectional reset timing.

Figure 200. SW / WDT unidirectional RESET ($\overline{EA}$ = 1)

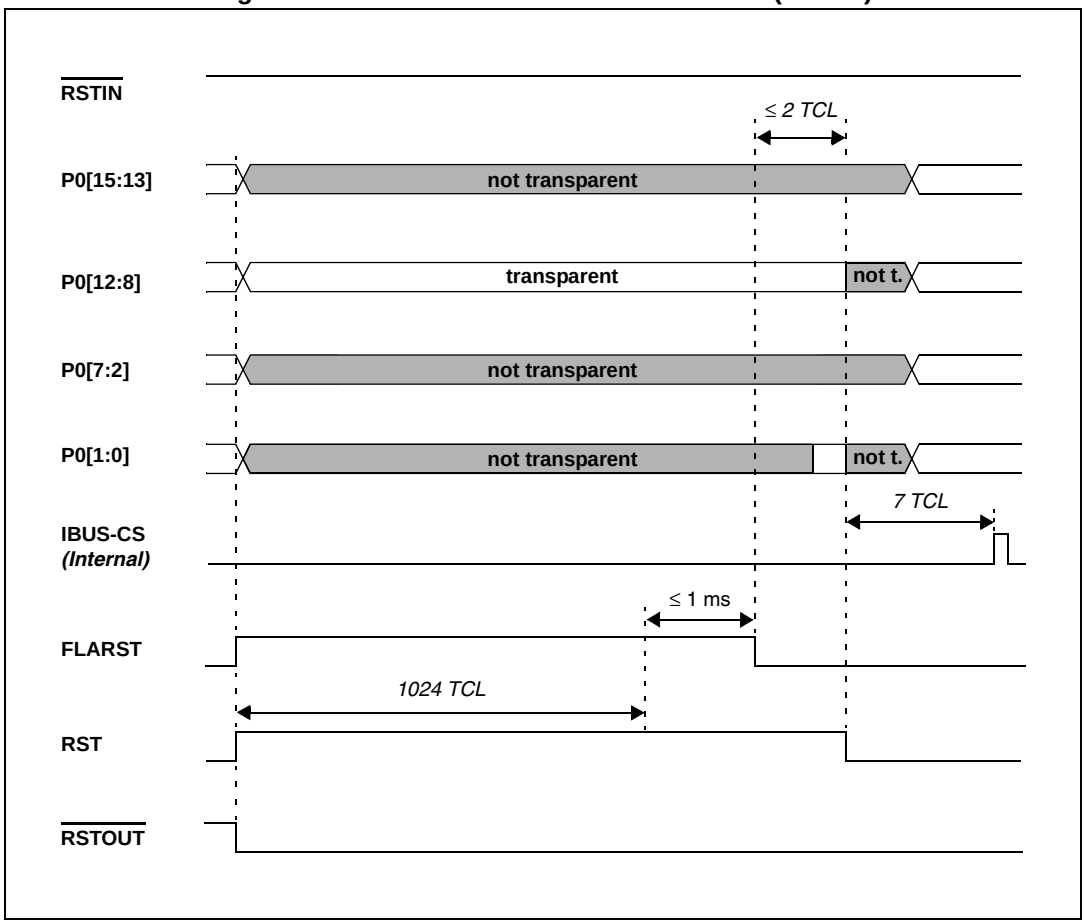
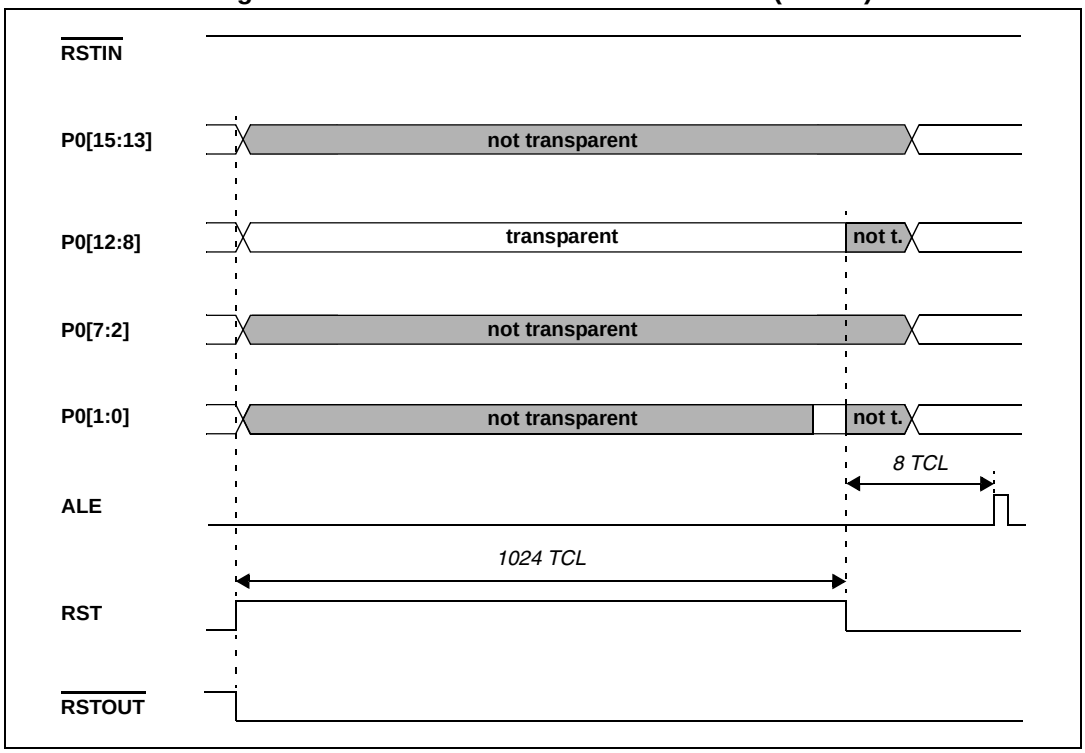


Figure 201. SW / WDT unidirectional RESET ($\overline{EA}$ = 0)



23.6 Bidirectional reset

As shown in the previous sections, the $\overline{RSTOUT}$ pin is driven active (low level) at the beginning of any reset sequence (synchronous/asynchronous hardware, software and watchdog timer resets). $\overline{RSTOUT}$ pin stays active low beyond the end of the initialization routine, until the protected EINIT instruction (End of Initialization) is completed.

The Bidirectional Reset function is useful when external devices require a reset signal but cannot be connected to $\overline{RSTOUT}$ pin, because $\overline{RSTOUT}$ signal lasts during initialization. It is, for instance, the case of external memory running initialization routine before the execution of EINIT instruction.

Bidirectional reset function is enabled by setting bit 3 (BDRSTEN) in SYSCON register. It only can be enabled during the initialization routine, before EINIT instruction is completed.

When enabled, the open drain of the $\overline{RSTIN}$ pin is activated, pulling down the reset signal, for the duration of the internal reset sequence (synchronous/asynchronous hardware, synchronous software and synchronous watchdog timer resets). At the end of the internal reset sequence the pull down is released and:

- After a Short Synchronous Bidirectional Hardware Reset, if $\overline{RSTF}$ is sampled low 8 TCL periods after the internal reset sequence completion (refer to *Figure 196* and *Figure 197*), the Short Reset becomes a Long Reset. On the contrary, if $\overline{RSTF}$ is sampled high the device simply exits reset state.
- After a Software or Watchdog Bidirectional Reset, the device exits from reset. If $\overline{RSTF}$ remains still low for at least 4 TCL periods (minimum time to recognize a Short Hardware reset) after the reset exiting (refer to *Figure 202* and *Figure 203*), the

Software or Watchdog Reset become a Short Hardware Reset. On the contrary, if RSTF remains low for less than 4 TCL, the device simply exits reset state.

The Bidirectional reset is not effective in case RPD is held low, when a Software or Watchdog reset event occurs. On the contrary, if a Software or Watchdog Bidirectional reset event is active and RPD becomes low, the RSTIN pin is immediately released, while the internal reset sequence is completed regardless of RPD status change (1024 TCL).

Note: The bidirectional reset function is disabled by any reset sequence (bit BDRSTEN of SYSCON is cleared). To be activated again it must be enabled during the initialization routine.

WDTCON flags

Similarly to what already highlighted in the previous section when discussing about Short reset and the degeneration into Long reset, similar situations may occur when Bidirectional reset is enabled. The presence of the internal filter on RSTIN pin introduces a delay: when RSTIN is released, the internal signal after the filter (see RSTF in the drawings) is delayed, so it remains still active (low) for a while. It means that depending on the internal clock speed, a short reset may be recognized as a long reset: the WDTCON flags are set accordingly.

Besides, when either Software or Watchdog bidirectional reset events occur, again when the RSTIN pin is released (at the end of the internal reset sequence), the RSTF internal signal (after the filter) remains low for a while, and depending on the clock frequency it is recognized high or low: 8TCL after the completion of the internal sequence, the level of RSTF signal is sampled, and if recognized still low a Hardware reset sequence starts, and WDTCON will flag this last event, masking the previous one (Software or Watchdog reset). Typically, a Short Hardware reset is recognized, unless the RSTIN pin (and consequently internal signal RSTF) is sufficiently held low by the external hardware to inject a Long Hardware reset. After this occurrence, the initialization routine is not able to recognize a Software or Watchdog bidirectional reset event, since a different source is flagged inside WDTCON register. This phenomenon does not occur when internal Flash is selected during reset (EA = 1), since the initialization of the Flash itself extend the internal reset duration well beyond the filter delay.

Figure 202, *Figure 203* and *Figure 204* summarize the timing for Software and Watchdog Timer Bidirectional reset events: In particular *Figure 204* shows the degeneration into Hardware reset.

Figure 202. SW / WDT bidirectional RESET($\overline{EA}$ = 1)

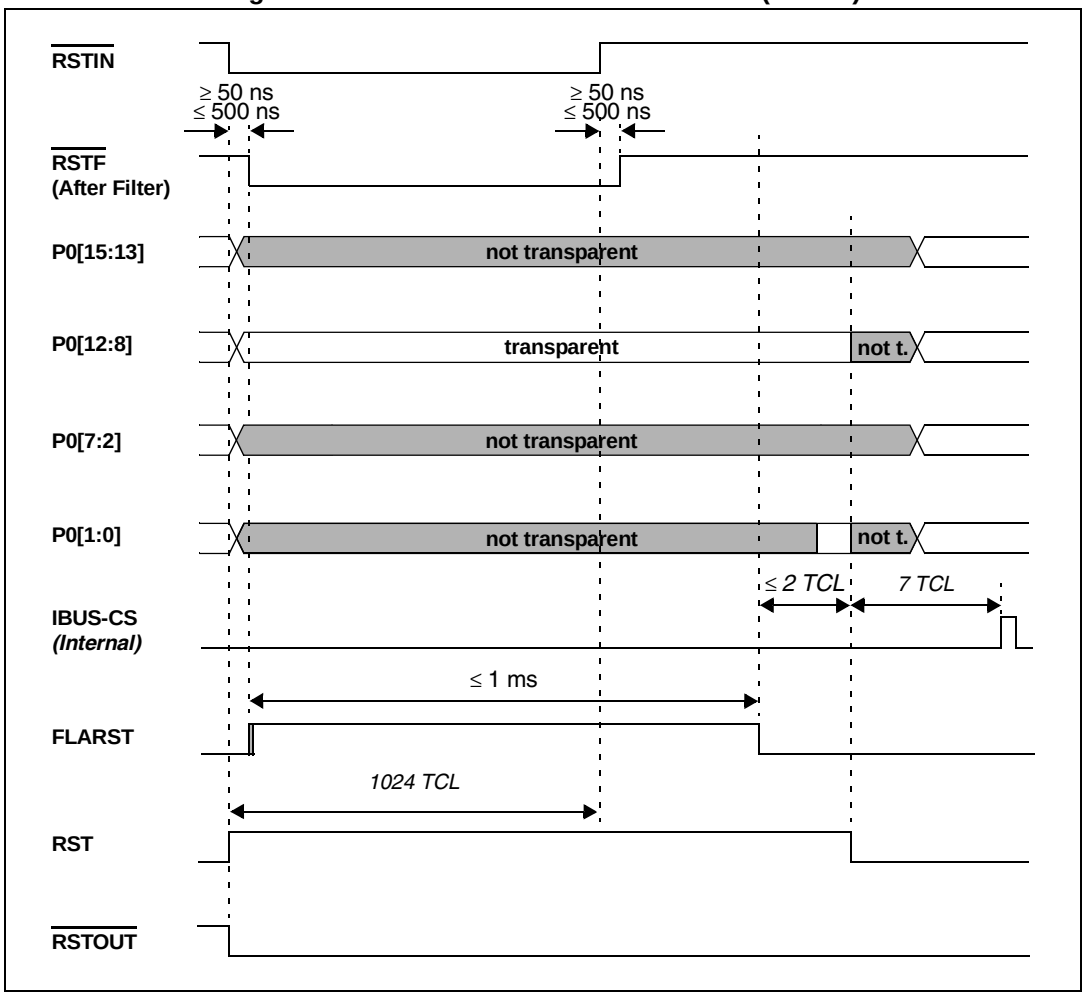


Figure 203. SW / WDT bidirectional RESET ($\overline{EA}$ = 0)

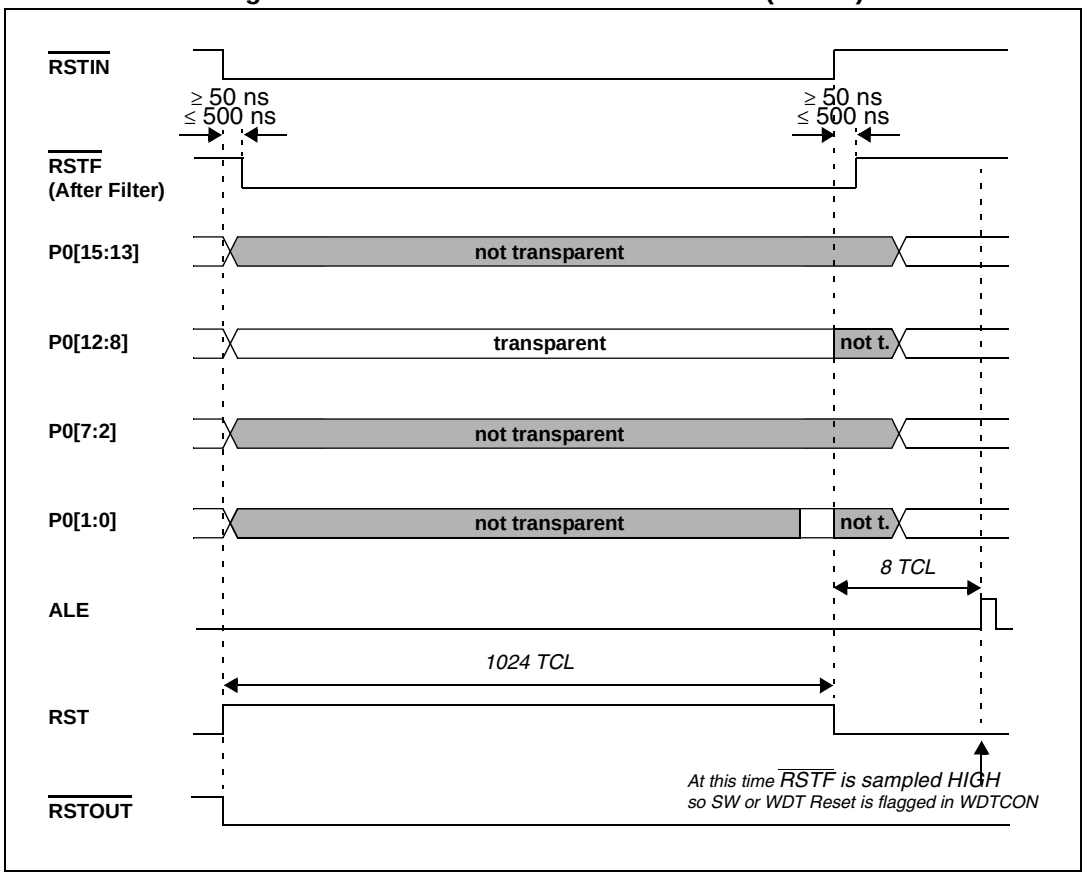
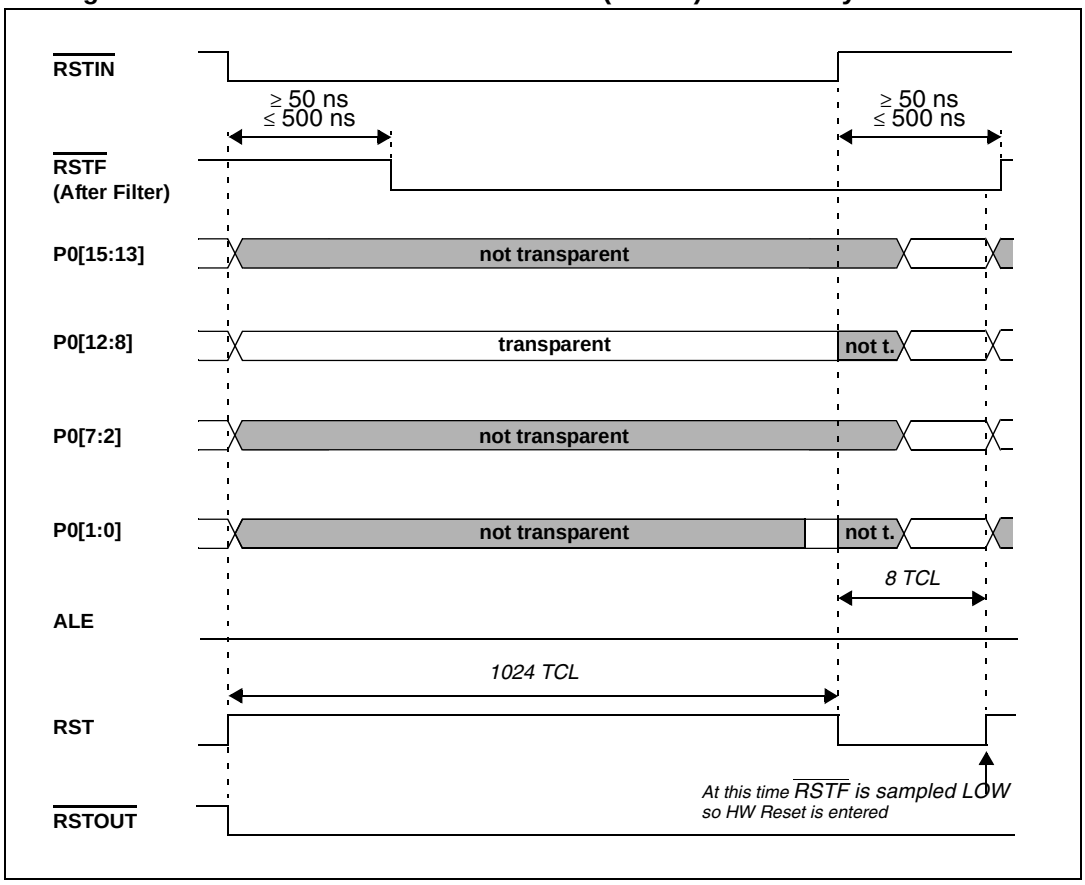


Figure 204. SW / WDT bidirectional RESET ($\overline{EA}$ = 0) followed by a HW RESET



23.7 Reset circuitry

Internal reset circuitry is described in *Figure 3 on page 30*. The $\overline{RSTIN}$ pin provides an internal pull-up resistor of 50kΩ to 250kΩ (The minimum reset time must be calculated using the lowest value).

It also provides a programmable (BDRSTEN bit of SYSCON register) pull-down to output internal reset state signal (synchronous reset, watchdog timer reset or software reset).

This bidirectional reset function is useful in applications where external devices require a reset signal but cannot be connected to $\overline{RSTOUT}$ pin.

This is the case of an external memory running codes before EINIT (end of initialization) instruction is executed. $\overline{RSTOUT}$ pin is pulled high only when EINIT is executed.

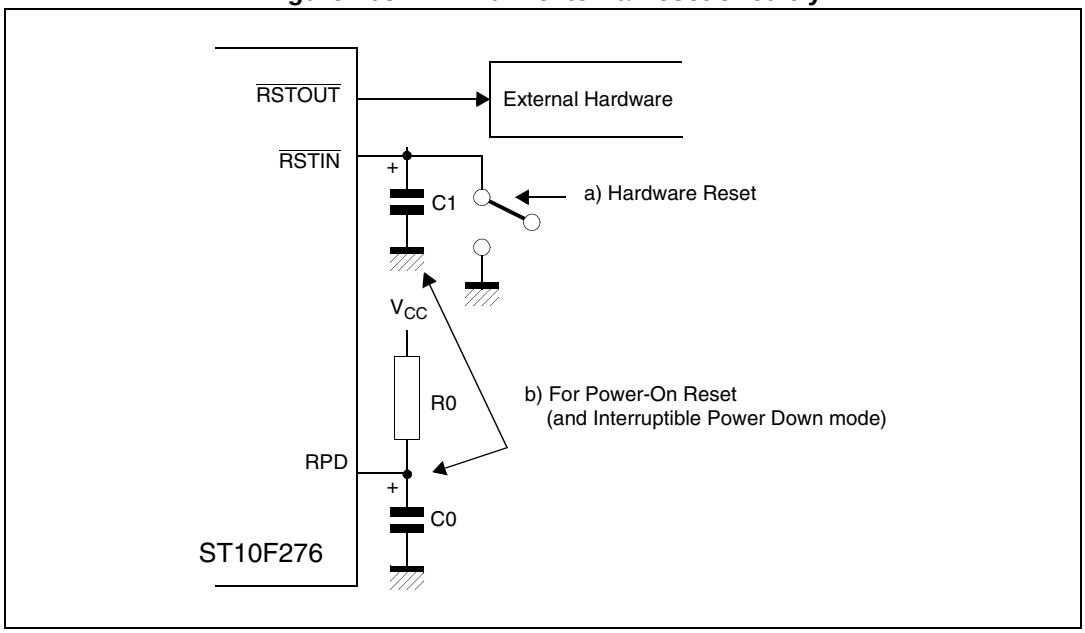
The RPD pin provides an internal weak pull-down resistor which discharges external capacitor at a typical rate of 200µA. If bit PWDCFG of SYSCON register is set, an internal pull-up resistor is activated at the end of the reset sequence. This pull-up will charge any capacitor connected on RPD pin.

The simplest way to reset the ST10F276 is to insert a capacitor C1 between $\overline{RSTIN}$ pin and V_{SS}, and a capacitor between RPD pin and V_{SS} (C0) with a pull-up resistor R0 between RPD pin and V_{DD}. The input $\overline{RSTIN}$ provides an internal pull-up device equalling a resistor of 50kΩ to 250kΩ (the minimum reset time must be determined by the lowest value). Select

C1 that produces a sufficient discharge time to permit the internal or external oscillator and / or internal PLL and the on-chip voltage regulator to stabilize.

To ensure correct Power-On reset with controlled supply current consumption, specially if clock signal requires a long period of time to stabilize, an asynchronous hardware reset is required during Power-On. For this reason, it is recommended to connect the external R0-C0 circuit shown in *Figure 205* to the RPD pin. At Power-On, the logical low level on RPD pin forces an asynchronous hardware reset when $\overline{\text{RSTIN}}$ is asserted low. The external pull-up R0 will then charge the capacitor C0. Note that an internal pull-down device on RPD pin is turned on when $\overline{\text{RSTIN}}$ pin is low, and causes the external capacitor (C0) to begin discharging at a typical rate of 100-200µA. With this mechanism, after Power-On reset, short low pulses applied on $\overline{\text{RSTIN}}$ produce synchronous hardware reset. If $\overline{\text{RSTIN}}$ is asserted longer than the time needed for C0 to be discharged by the internal pull-down device, then the device is forced in an asynchronous reset. This mechanism insures recovery from very catastrophic failure.

Figure 205. Minimum external reset circuitry

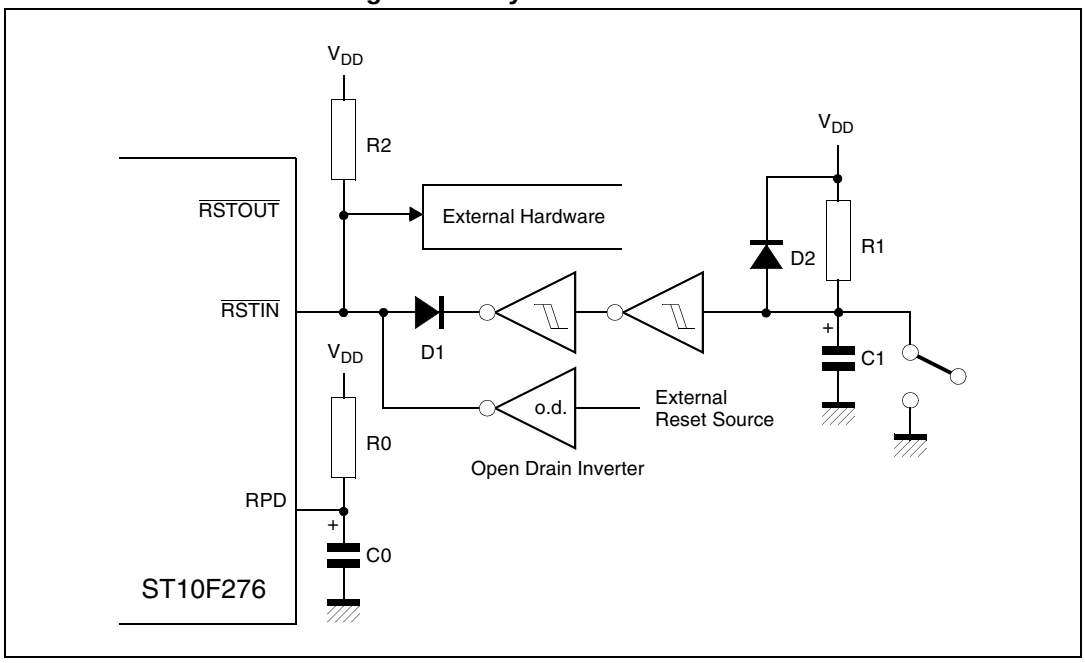


The minimum reset circuit of *Figure 205* is not adequate when the $\overline{\text{RSTIN}}$ pin is driven from the ST10F276 itself during software or watchdog triggered resets, because of the capacitor C1 that will keep the voltage on $\overline{\text{RSTIN}}$ pin above V_{IL} after the end of the internal reset sequence, and thus will trigger an asynchronous reset sequence.

Figure 206 shows an example of a reset circuit. In this example, R1-C1 external circuit is only used to generate Power-On or manual reset, and R0-C0 circuit on RPD is used for Power-On reset and to exit from Power Down mode. Diode D1 creates a wired-OR gate connection to the reset pin and may be replaced by open-collector Schmitt Trigger buffer. Diode D2 provides a faster cycle time for repetitive Power-On resets.

R2 is an optional pull-up for faster recovery and correct biasing of TTL Open Collector drivers.

Figure 206. System reset circuit



23.8 Reset application examples

Next two timing diagrams (*Figure 207* and *Figure 208*) provides additional examples of bidirectional internal reset events (Software and Watchdog) including in particular the external capacitances charge and discharge transients (refer also to *Figure 206* for the external circuit scheme).

Figure 207. Example of software or watchdog bidirectional reset ($\overline{EA}$ = 1)

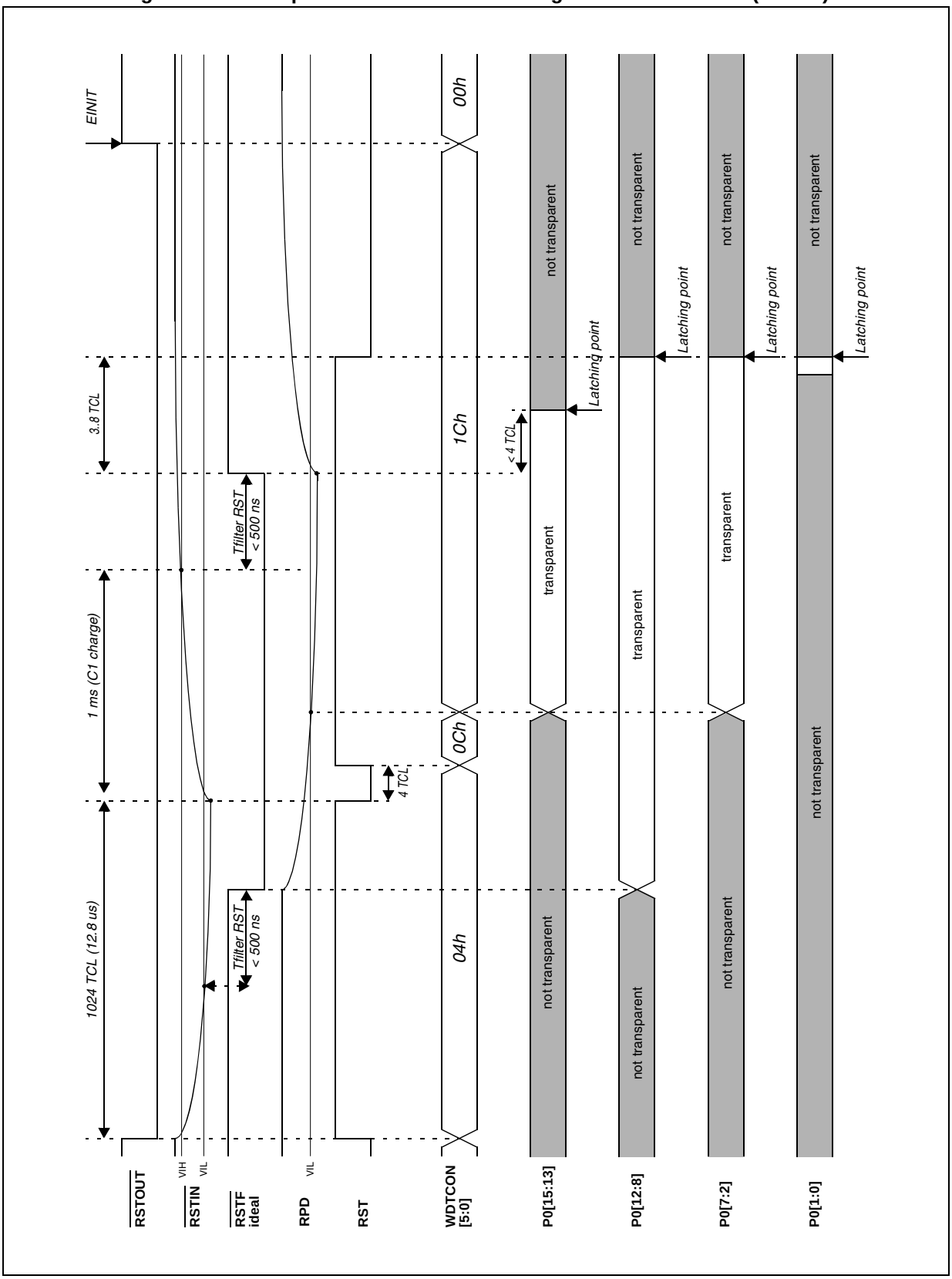
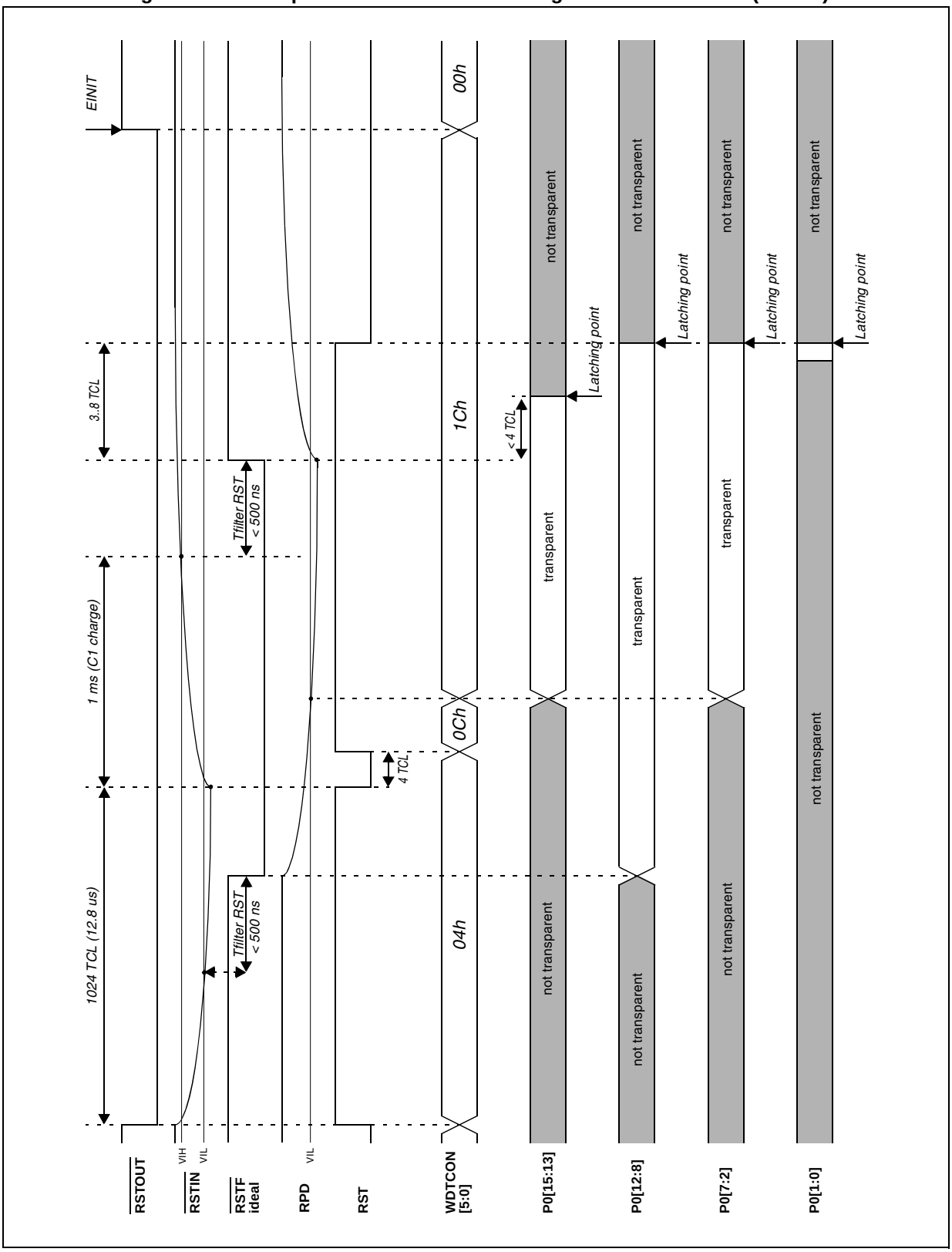


Figure 208. Example of software or watchdog bidirectional reset ($\overline{EA}$ = 0)



23.9 Reset summary

A summary of the different reset events is reported in the table below.

Table 64. Reset events summary

Event	RPD	EA	Bidir	Synch. asynch.	RSTIN		WDTCON flags				
					min	max	PONR	LHWR	SHWR	SWR	WDTR
Power-on reset	0	0	N	Asynch.	1 ms (VREG) 1.2 ms (Reson. + PLL) 10.2 ms (Crystal + PLL)	-	1	1	1	1	0
	0	1	N	Asynch.	1ms (VREG)	-	1	1	1	1	0
	1	x	x	Forbidden							
	x	x	Y	Not applicable							
Hardware reset (asynchronous)	0	0	N	Asynch.	500ns	-	0	1	1	1	0
	0	1	N	Asynch.	500ns	-	0	1	1	1	0
	0	0	Y	Asynch.	500ns	-	0	1	1	1	0
	0	1	Y	Asynch.	500ns	-	0	1	1	1	0
Short hardware reset (synchronous) (1)	1	0	N	Synch.	max (4 TCL, 500ns)	1032 + 12 TCL + max(4 TCL, 500ns)	0	0	1	1	0
	1	1	N	Synch.	max (4 TCL, 500ns)	1032 + 12 TCL + max(4 TCL, 500ns)	0	0	1	1	0
	1	0	Y	Synch.	max (4 TCL, 500ns)	1032 + 12 TCL + max(4 TCL, 500ns)	0	0	1	1	0
					Activated by internal logic for 1024 TCL						
	1	1	Y	Synch.	max (4 TCL, 500ns)	1032 + 12 TCL + max(4 TCL, 500ns)	0	0	1	1	0
					Activated by internal logic for 1024 TCL						
Long hardware reset (synchronous)	1	0	N	Synch.	1032 + 12 TCL + max(4 TCL, 500ns)	-	0	1	1	1	0
	1	1	N	Synch.	1032 + 12 TCL + max(4 TCL, 500ns)	-	0	1	1	1	0
	1	0	Y	Synch.	1032 + 12 TCL + max(4 TCL, 500ns)	-	0	1	1	1	0
					Activated by internal logic only for 1024 TCL						
	1	1	Y	Synch.	1032 + 12 TCL + max(4 TCL, 500ns)	-	0	1	1	1	0
					Activated by internal logic only for 1024 TCL						

Table 64. Reset events summary (continued)

Event	RPD	EA	Bidir	Synch. asynch.	RSTIN		WDTCON flags				
					min	max	PONR	LHWR	SHWR	SWR	WDTR
Software reset (2)	x	0	N	Synch.	Not activated		0	0	0	1	0
	x	0	N	Synch.	Not activated		0	0	0	1	0
	0	1	Y	Synch.	Not activated		0	0	0	1	0
	1	1	Y	Synch.	Activated by internal logic for 1024 TCL		0	0	0	1	0
Watchdog reset (2)	x	0	N	Synch.	Not activated		0	0	0	1	1
	x	0	N	Synch.	Not activated		0	0	0	1	1
	0	1	Y	Synch.	Not activated		0	0	0	1	1
	1	1	Y	Synch.	Activated by internal logic for 1024 TCL		0	0	0	1	1

1. It can degenerate into a Long Hardware Reset and consequently differently flagged (see *Section 24.3* for details).
2. When Bidirectional is active (and with RPD = 0), it can be followed by a Short Hardware Reset and consequently differently flagged (see *Section 23.6: Bidirectional reset* for details).

23.9.1 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 $\overline{EA}$).

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, 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 ST10F276.

Not all PORT0 bits are latched after the end of an internal reset. Depending on the reset type, different bits are latched.

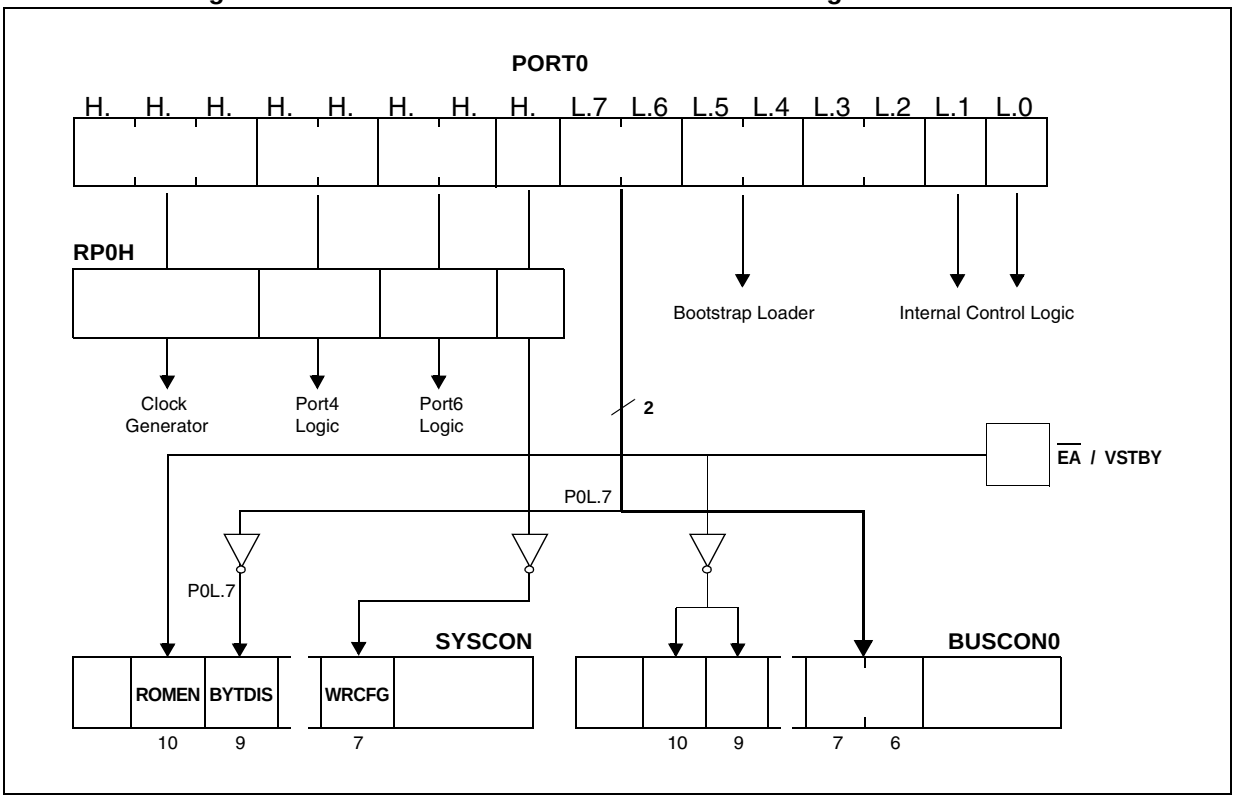
When $\overline{RSTIN}$ goes active, the PORT0 configuration input pins are not transparent for the first 1024 TCL.

After that time only, the PORT0 pins are transparent and will be latched when internal reset signal becomes inactive (see falling edge of internal signal RST in *Figure 192 on page 474* to *Figure 199 on page 483*). To avoid unexpected behavior, the level of the PORT0 configuration input pins should not change while PORT0 is transparent.

Table 65. PORT0 latched configuration for the different reset events

X : Pin is sampled - : Pin is not sampled	PORT0															
	Clock Options			Segm. Addr. Lines		Chip Selects		WR config.	Bus Type		Reserved	BSL	Reserved	Reserved	Adapt Mode	Emu Mode
Sample event	P0H.7	P0H.6	P0H.5	P0H.4	P0H.3	P0H.2	P0H.1	P0H.0	P0L.7	P0L.6	P0L.5	P0L.4	P0L.3	P0L.2	P0L.1	P0L.0
Software Reset	-	-	-	X	X	X	X	X	X	X	-	-	-	-	-	-
Watchdog Reset	-	-	-	X	X	X	X	X	X	X	-	-	-	-	-	-
Synchronous Short Hardware Reset	-	-	-	X	X	X	X	X	X	X	X	X	X	X	X	X
Synchronous Long Hardware Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Asynchronous Hardware Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Asynchronous Power-On Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 209. PORT0 bits latched into the different registers after reset



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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	CLKCFG			SALSEL		CSSEL		WRC
								R			R		R		R

Bit	Function
WRC(1)	**Write Configuration Control** '0': Pins $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$ '1': Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function
CSSEL(1)	**Chip Select Line Selection** (Number of active $\overline{CS}$ outputs) 0 0: 3 $\overline{CS}$ lines: $\overline{CS}$2...$\overline{CS}$0 0 1: 2 $\overline{CS}$ lines: $\overline{CS}$1...$\overline{CS}$0 1 0: No $\overline{CS}$ lines at all 1 1: 5 $\overline{CS}$ lines: $\overline{CS}$4...$\overline{CS}$0 (Default without pull-downs)
SALSEL(1)	**Segment Address Line Selection** (Number of active segment address outputs) 0 0: 4 Bit segment address: A19...A16 0 1: No segment address lines at all 1 0: 8 Bit segment address: A23...A16 1 1: 2 Bit segment address: A17...A16 (Default without pull-downs)
CLKCFG(1) (2)	P0H.7-5 $f_{CPU} = f_{XTAL} \times F$ Notes (3) 111 $f_{XTAL} \times 4$ Default configuration 110 $f_{XTAL} \times 3$ 101 $f_{XTAL} \times 8$ 100 $f_{XTAL} \times 5$ 011 f_{XTAL} Direct Drive (oscillator bypassed) (3) 010 $f_{XTAL} \times 10$ 001 $f_{XTAL} \times 0.5$ CPU clock via prescaler (3) 000 $f_{XTAL} \times 16$

1. These bits are set according to Port0 configuration during any reset sequence.
2. RP0H.7 to RP0H.5 bits are loaded only during a long hardware reset. As pull-up resistors are active on each Port P0H pins during reset, RP0H default value is 'FFh'.
3. Refer to datasheet for more details about input clock ranges and limitations.

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 ST10F276 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 *Section 8: The external bus interface on page 181*.

Note: The reserved pins, P0L.2 and P0L.3, must remain high during reset in order to ensure proper operation of the ST10F276. 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: P0L.0

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 ST10F276. In addition also the RSTOUT pin floats to tristate rather to be driven low. When the emulation mode is latched the CLKOUT output is automatically enabled. This mode is used for special emulator purposes and is not used in basic ST10F276 devices, so in this case P0L.0 should be held high.

Default: Emulation Mode is off.

Adapt mode: P0L.1

Pin P0L.1 (ADP) selects the Adapt Mode, when low during reset. In this mode the ST10F276 goes into a passive state, which is similar to its state during reset.

The pins of the ST10F276 float to tristate or are deactivated via internal pull-up/pull-down devices, as described for the reset state. In addition also the RSTOUT pin floats to tristate rather than be driven low, and the on-chip oscillator is switched off.

This mode allows switching a ST10F276 that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original ST10F276 remains in its place. The original ST10F276 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).

Reserved: P0L.2 - P0L.3

These pins must be always left open (or driven high) during reset, to avoid ST10 enters particular test mode and consequently does not work properly.

Default: Test Modes are off.

Bootstrap loader mode: P0L.4 - P0L.5

Pins P0L.4 and P0L.5 (BSL) activate the on-chip bootstrap loader modes. The bootstrap loader allows moving the start code into the IRAM of the ST10F276 via the serial interface ASC0 or CAN1. The MCU will remain in bootstrap loader mode until a hardware reset with P0L.4 and P0L.5 both high or a software reset occurrence. Refer to *Section 15: The bootstrap loader on page 302* for details.

Default: The ST10F276 starts fetching code from location 00'0000h, the bootstrap loader is off.

External bus type: P0L.6 - P0L.7

Pins P0L.7 and P0L.6 (BUSTYP) select the external bus type during reset, if an external start is selected via pin EA. This allows the configuration of the external bus interface of the

ST10F276 even for the first code fetch after reset. The two bits are copied into bit-field BTYP of register BUSCON0. P0L.7 controls the data bus width, while P0L.6 controls the address output (multiplexed or de-multiplexed). This bit-field may be changed via software after reset, if required.

BTYP encoding	External data bus width	External address bus mode
0 0	8-bit Data	De-multiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	De-multiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

PORT0 and PORT1 are automatically switched to the selected bus mode. In multiplexed bus modes PORT0 drives both the 16-bit intra-segment address and the output data, while PORT1 remains in high impedance state as long as no de-multiplexed bus is selected via one of the BUSCON registers. In de-multiplexed bus modes PORT1 drives the 16-bit intra-segment address, while PORT0 or P0L (according to the selected data bus width) drives the output data.
For a 16-bit data bus $\overline{BHE}$ is automatically enabled, for an 8-bit data bus $\overline{BHE}$ is disabled via bit BYTDIS in register SYSCON.

Default: 16-bit data bus with multiplexed addresses.

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

Write configuration: P0H.0

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

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

Chip select lines: P0H.1 - P0H.2

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

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

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

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

Note: The selected number of $\overline{CS}$ signals cannot be changed via software after reset.

Segment address lines: P0H.3 - P0H.4

Pins P0H.4 and P0H.3 (SALSEL) define the number of active segment address lines during reset. This determines which pins of Port4 are used as address line or as I/O line. The two bits are latched in register RP0H.

Depending on the system architecture the required address space is chosen and accessible right from the start, so the initialization routine can directly access all locations without prior programming.

The required pins of Port4 are automatically switched to address output mode.

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

Even if not all segment address lines are enabled on Port4, the ST10F276 internally uses its complete 24-bit addressing mechanism.

This allows the restriction of the width of the effective address bus, while still deriving $\overline{CS}$ signals from the complete addresses.

Default: 2-bit segment address (A17...A16) allowing access to 256 Kbytes.

Note: The selected number of segment address lines cannot be changed via software after reset.

Clock generation control: P0H.5 - P0H.6 - P0H.7

Pins P0H.7, P0H.6 and P0H.5 (CLKCFG) select the clock generation mode (on-chip PLL) during reset. The oscillator clock either directly feeds the CPU and peripherals (direct drive) or it is fed to the on-chip PLL which then provides the CPU clock signal (selectable multiple of the oscillator frequency). These bits are latched in register RP0H (see previous *Section 23.9.1: System start-up configuration on page 496*).

24 Power reduction modes

Several different power reduction modes with different levels of power reduction have been implemented in the ST10F276, which may be entered under software and/or hardware control.

In **Idle mode** the CPU is stopped, while the peripherals continue their operation. Idle mode can be terminated by any reset or interrupt request.

In **Power Down mode** both the CPU and the peripherals are stopped. Power Down mode can be configured by software in order to be terminated only by a hardware reset, by a transition on enabled fast external interrupt pins, by an interrupt generated by the Real Time Clock, by an interrupt generated by the activity on CAN's and I^2C module interfaces.

*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 for negative polarity, or driven high for positive polarity) during the last bus access.*

When Real Time Clock module is used, when the device is in Power Down mode a reference clock is needed. In this case, two possible configurations may be selected by the user application according to the desired level of power reduction:

- A 32 kHz crystal is connected to the low-power oscillator pins (XTAL3 / XTAL4) and running. In this case the main oscillator is stopped when Power Down mode is entered, while the Real Time Clock continues counting using 32 kHz clock signal as reference. The presence of a running low-power oscillator is detected after the Power-On: this clock is immediately assumed (if present, or as soon as it is detected) as reference for the Real Time Clock counter and it will be maintained forever (unless specifically disabled via software, see *Section 22: Real time clock on page 464*).
- Only the main oscillator is running (XTAL1 / XTAL2 pins). In this case the main oscillator is not stopped when Power Down is entered, and the Real Time Clock continues counting using the main oscillator clock signal as reference.

Stand-by mode is achieved by turning off the main power supply (V_{DD}) while V_{STBY} remains the only active supply for the device. In this condition V_{STBY} pin provides the supply to a portion of the XRAM (the so called Stand-by RAM, 16 Kbyte in this device) through a dedicated on-chip low power Voltage Regulator: the content of this RAM can be retained and will be available at next system start-up.

Note: V_{STBY} should be always powered in the range of 4.5-5.5 volt: 6 volt is acceptable for a reduced period of time during the life of the device (refer to Electrical Characteristics section for details); 4 volt is acceptable when no RTC and 32 kHz oscillator are used.

Note: Exception for V_{STBY} value is allowed when $\overline{RSTIN}$ pin is held low and the main V_{DD} is on: this will allow to properly drive pin $\overline{EA}$ (mapped together with V_{STBY}) and configure the access to external memory. After $\overline{RSTIN}$ pin is released, V_{STBY} should return high, to be used as V_{STBY} supply voltage.

When Real Time Clock module is counting, it is still possible to enter Stand-by Mode. In particular, when the on-chip low-power oscillator (XTAL3 / XTAL4 pins) provides a 32 kHz clock reference for the counter, the main power supply can be turned off since both Real Time Clock module and the low-power oscillator circuitries are powered (directly or through the low-power voltage regulator) by the voltage applied on V_{STBY} pin.

Note: If the on-chip low-power oscillator is not in use, the Real Time Clock module can only be driven by the main oscillator: in this case it is not possible to turn off the main power supply (V_{DD}) of the device, since the main oscillator circuitry would stop working. In this case, standard Power Down mode is recommended (V_{DD} on, so RTC can be driven by the main oscillator).

Table 66. Power reduction modes summary

Mode	V_{DD}	V_{STBY}	CPU	Peripherals	RTC	Main OSC	32 kHz OSC	STBY XRAM	XRAM
Idle	on	on	off	on	off	on	off	biased	biased
	on	on	off	on	on	on	on	biased	biased
	on	on	off	on	on	on	off	biased	biased
Power Down	on	on	off	off	off	off	off	biased	biased
	on	on	off	off	on	on	off	biased	biased
	on	on	off	off	on	off	on	biased	biased
Stand-by	off	on	off	off	off	off	off	biased	off
	off	on	off	off	on	off	on	biased	off

24.1 Idle mode

The power consumption of the ST10F276 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.

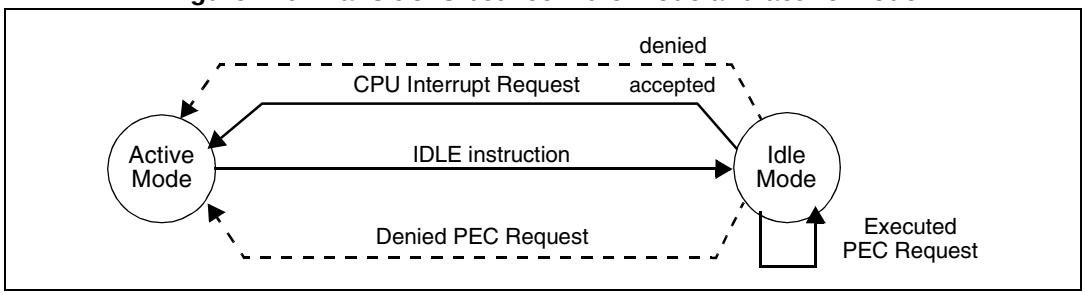
For a request selected for CPU interrupt service the associated interrupt service routine is entered if the priority level of the requesting source is higher than the current CPU priority and the interrupt system is globally enabled. After the RETI (Return from Interrupt) instruction of the interrupt 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 210*).

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 210. 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.

24.2 Power down mode

To further reduce the power consumption the microcontroller can be switched to Power Down mode. Clocking of all internal blocks is stopped, the contents of the on-chip RAM modules, however, are preserved through the voltage supplied via the V_{DD} pins (and on-chip voltage regulator). The watchdog timer is stopped in Power Down mode also. The only exception could be the Real Time Clock if opportunely programmed and one of the two oscillator circuits as a consequence (either the main or the low-power on-chip oscillator).

Before entering Power Down mode (by executing the instruction PWRDN), bit VREGOFF in XMISC register must be set: in this way, as soon as the PWRDN command is executed, the main voltage regulator is turned off, and only the so called low power voltage regulator remains active.

Note: Leaving the main voltage regulator active during Power Down may lead to unexpected behavior (ex: CPU wake-up) and power consumption higher than what specified.

XMISC (EB46h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–												VREG OFF	CAN CK2	CAN PAR	ADC MUX
–												RW	RW	RW	RW

Bit	Function
ADCMUX	**Port1L ADC Channels Enable** '0': Analog inputs on port P5.y can be converted (default configuration) '1': Analog inputs on port P1.z can be converted. Only 8 channels can be managed
CANPAR	**CAN Parallel Mode Selection** '0':CAN2 is mapped on P4.4/P4.7, while CAN1 is mapped on P4.5/P4.6 '1':CAN1 and CAN2 are mapped in parallel on P4.5/P4.6. This is effective only if both CAN1 and CAN2 are enabled through setting of bits CAN1EN and CAN2EN in XPERCON register. If CAN1 is disabled, CAN2 remains on P4.4/P4.7 even if bit CANPAR is set.
CANCK2	**CAN Clock divider by 2 disable** '0':Clock provided to CAN modules is CPU clock divided by 2 (mandatory when f_{CPU} is higher than 40 MHz) '1': Clock provided to CAN modules is directly CPU clock
VREGOFF	**Main Voltage Regulator disable for Power Down mode** '0': Default value after reset and when Power Down is not used. '1': On-chip Main Regulator is turned off when Power Down mode is entered.

The ST10F276 provides two different operating Power Down modes:

- Protected Power Down mode,
- Interruptible Power Down mode.

The Power Down operating mode is selected by the bit PWDCFG in SYSCON register.

SYSCON (FF12h / 89h) SFR Reset Value: 0xx0h

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

Reset Value: 0000 0xx0 x000 0000b

Bit	Function
PWDCFG	**Power Down Mode Configuration Control** '0': Power Down Mode can only be entered during PWRDN instruction execution if NMI pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must be provided 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 and/or by an interrupt coming from the Real Time Clock (if running), and/or by an interrupt coming from CAN1/CAN2/I^2C serial interfaces and/or by asserting RSTIN pin.

Note: Register SYSCON cannot be changed after execution of the EINIT instruction.

24.2.1 Protected power down mode

This mode is selected by clearing the bit PWDCFG in register SYSCON to '0'.

In this mode, the Power Down mode can **only** be entered 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 which can save the internal state into RAM. After the internal state has been saved, the trap routine may set a flag or write a certain bit pattern into specific RAM locations, and then execute the PWRDN instruction. If the NMI pin is still low at this time, Power Down mode will be entered, otherwise program execution continues.

Exiting power down mode

In this mode, the **only** way to exit Power Down mode is with an external hardware reset.

The initialization routine (executed upon reset) can check the identification flag (see WDTCON - *Section 14: Watchdog timer on page 297*) or bit pattern within RAM to determine whether the controller was initially switched on, or whether it was properly restarted from Power Down mode.

24.2.2 Interruptible power down mode

This mode is selected by setting the bit PWDCFG in register SYSCON to '1'.

In this mode, the Power Down mode can be entered if enabled Fast External Interrupt pins (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 follow:

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

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

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

Exiting power down mode

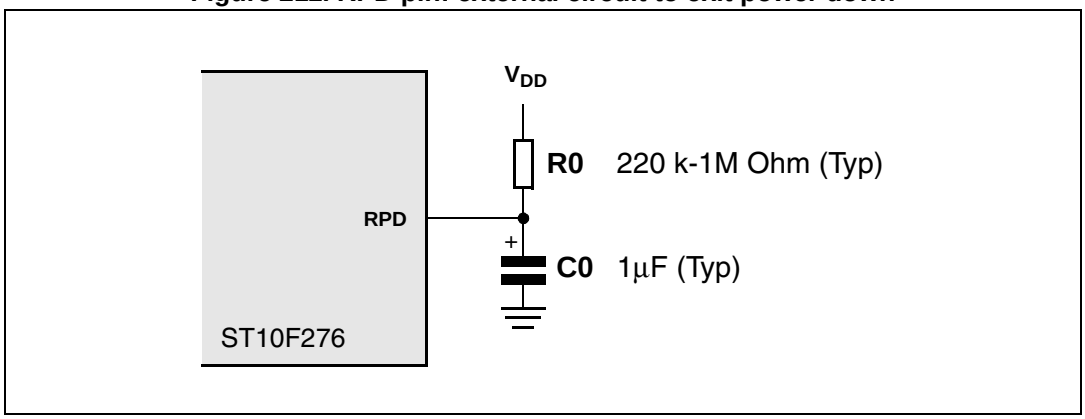
When Interruptible Power Down mode is entered, the CPU and peripheral clocks are frozen, and the oscillator and PLL are stopped (when RTC is disabled, so there is no need for a clock reference). Interruptible Power Down mode can be exited by either asserting $\overline{\text{RSTIN}}$ or one of the enabled EXxIN pin (Fast External Interrupt). In case the Real Time Clock module needs to be running during Power Down, either the main or the low-power oscillator is not stopped. The PLL, on the contrary is anyway switched off.

If Power Down mode is exited by a hardware RESET, $\overline{\text{RSTIN}}$ pin must be held low until the oscillator (if not already running for Real Time Clock operation) and PLL have restarted and stabilized.

EXxIN inputs are normally sampled interrupt inputs. However, the Power Down mode circuitry uses them as level-sensitive inputs. An EXxIN (x = 7...0) Interrupt Enable bit (bit CCxIE in respective CCxIC register) needs not to be set to bring the device out of Power Down mode.

In order to guarantee a proper stabilization time before restart the operation when exiting from Power Down (especially if the main oscillator was stopped: typically when Real Time Clock module is not used, or when the low-power on-chip oscillator circuit is used to provide the reference signal to the Real Time Clock module), an external RC circuit must be connected to RPD pin (Return from Power Down), as shown in the following *Figure 211*.

Figure 211. RPD pin: external circuit to exit power down



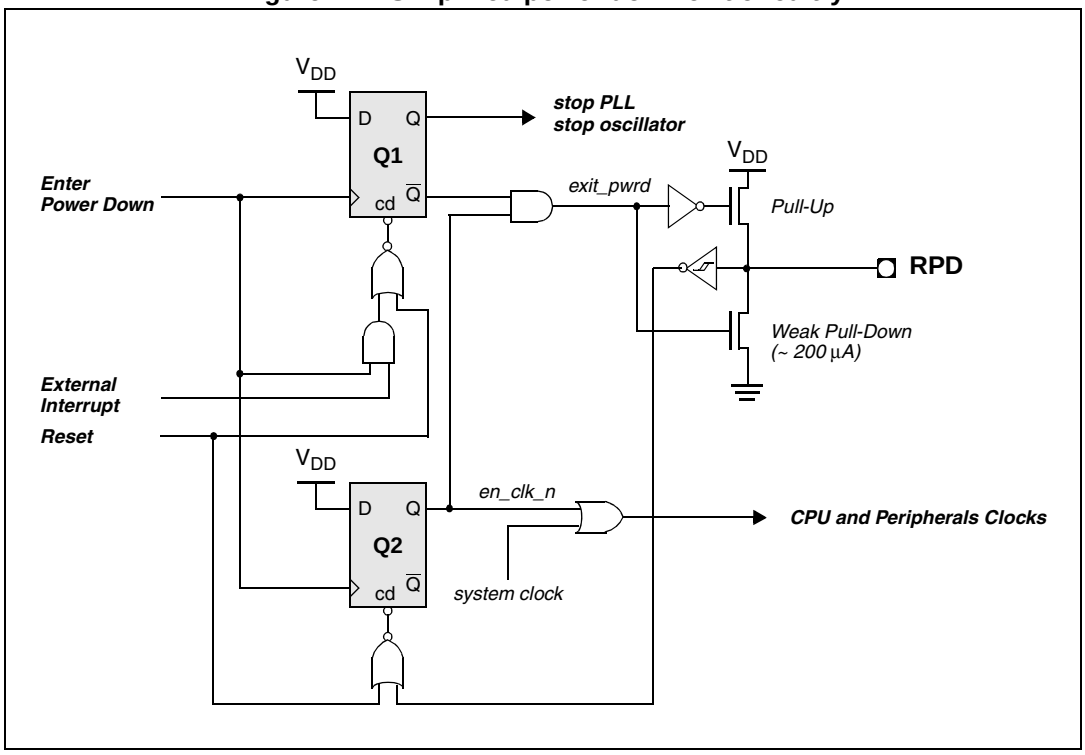
To exit Power Down mode with external interrupt, an EXxIN pin has to be asserted for at least 40ns (x = 7...0). This signal enables the internal main oscillator (if not already running) and PLL circuitry, and also turns on the internal weak pull-down on RPD pin (see following

Figure 212). 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 voltage on RPD pin drops below the threshold voltage (about 2.5 V), the Schmitt Trigger clears Q2 flip-flop, thus enabling the CPU and Peripheral clocks, and 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.

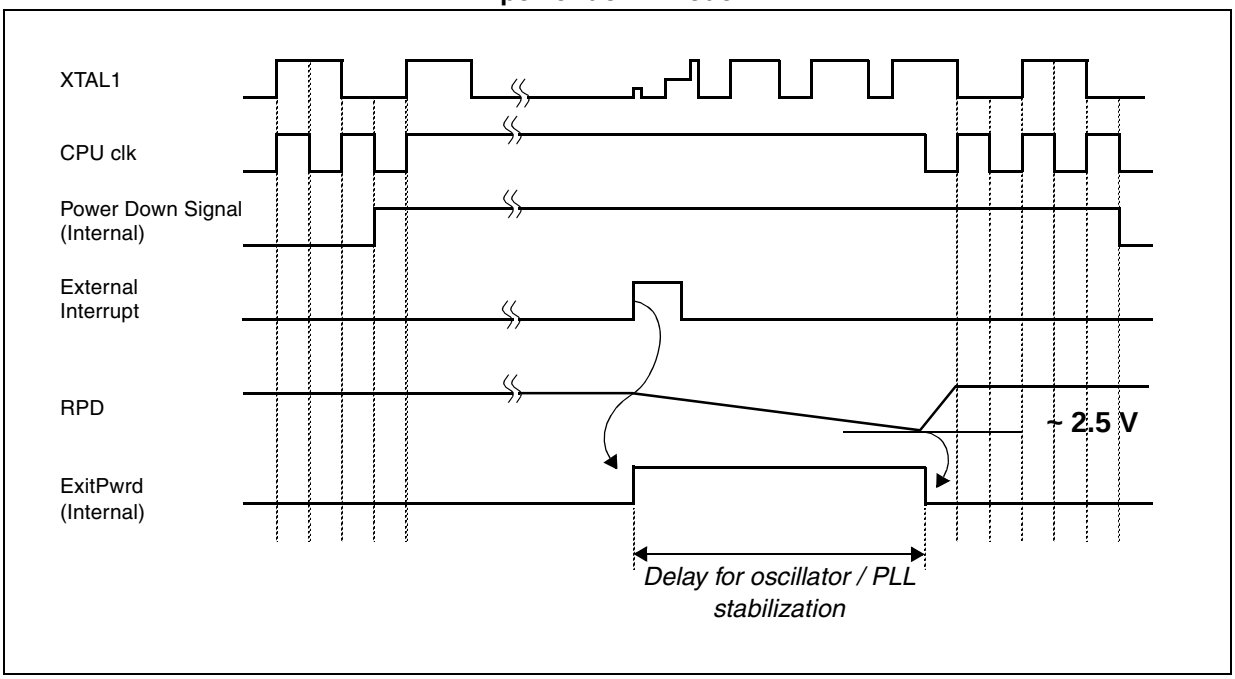
Note: Due to internal pipeline, the instruction that follows the PWRDN instruction is executed before the CPU performs a call of the interrupt service routine when exiting Power Down mode.

Figure 212. Simplified power down exit circuitry



Exiting from Interruptible Power Down is also possible through the CAN Receive lines and I^2C Serial Clock line: if properly enabled (through CC8IC and CC9IC registers), an activity on pins P4.5 and P4.4 is interpreted as a fast external interrupt event able to wake-up the device. For more details refers also to *Section 5.6.1: Fast external interrupts on page 115*.

Figure 213. Power down exit sequence using an external interrupt (PLL x 2) real time clock and power down mode



If the Real Time Clock is running (RTOFF bit of RTCCON register cleared), when PWRDN instruction is executed, the oscillator circuit which is providing the reference to the counter is not stopped. The selection of which of the two on-chip oscillator amplifier circuits should provide the reference clock to the Real Time Clock counter is determined whenever a Power-On sequence is applied on main V_{DD} pins. By default after Power-On, the reference clock is the main oscillator; immediately after exiting from Power-On reset ($\overline{\text{RSTIN}}$ pin released), an internal mechanism is able to detect the presence or absence of a clock signal from the low-power oscillator (32 kHz, see XTAL3 / XTAL4 pins). In case this oscillator is running, the reference for the Real Time Clock module becomes the 32 kHz ones (three to five 32 kHz clock pulses later). On the contrary, if no oscillation is detected on XTAL3 / XTAL4, the reference for the counter will remain the one from the main on-chip oscillator (XTAL1 / XTAL2).

Note: 1 In case the switch occurred (32 kHz reference selected), it is possible to come back to the default configuration either via software by setting the bit OFF32 of RTCCON register or through a new Power-On sequence.

2 When no external 32 kHz crystal is connected, XTAL3 and XTAL4 pins should be properly biased to a steady value, to avoid any spike, that would cause an unwanted switching of the reference clock for the Real Time Clock counter, from the main oscillator (running) signal to the low-power oscillator (not running, only noisy).

24.3 Stand-by mode

In Stand-by mode, the RAM array is maintained powered through the dedicated pin V_{STBY} when ST10F276 main power supply (V_{DD}) is turned off.

To enter Stand-by Mode is mandatory to held the device under reset: once the device is under reset, the RAM is disabled (see XRAM2EN bit of XPERCON register), and its digital

interface is frozen in order to avoid any kind of data corruption. It is then possible to turn off the main V_{DD} provided that V_{STBY} is on.

A dedicated embedded low-power voltage regulator is implemented to generate the internal low voltage supply (about 1.65V in Stand-by mode) to bias all those circuits that should remain active: the portion of XRAM (16 Kbytes), the RTC counters and 32 kHz on-chip oscillator amplifier.

In normal running mode (that is when main V_{DD} is on) the V_{STBY} pin can be tied to V_{SS} during reset to exercise the $\overline{EA}$ functionality associated with the same pin: the voltage supply for the circuitries which are usually biased with V_{STBY} (see in particular the 32 kHz oscillator used in conjunction with Real Time Clock module), is granted by the active main V_{DD}.

It must be noted that Stand-by Mode can generate problems associated with the usage of different power supplies in CMOS systems; particular attention must be paid when the ST10F276 I/O lines are interfaced with other external CMOS integrated circuits: if V_{DD} of ST10F276 becomes (for example in Stand-by Mode) lower than the output level forced by the I/O lines of these external integrated circuits, the ST10F276 could be directly powered through the inherent diode existing on ST10F276 output driver circuitry. The same is valid for ST10F276 interfaced to active/inactive communication buses during Stand-by mode: current injection can be generated through the inherent diode.
Furthermore, the sequence of turning on/off of the different voltage could be critical for the system (not only for the ST10F276 device). The device Stand-by mode current (I_{STBY}) may vary while V_{DD} to V_{STBY} (and vice versa) transition occurs: some current flows between V_{DD} and V_{STBY} pins. System noise on both V_{DD} and V_{STBY} can contribute to increase this phenomenon.

24.3.1 Entering stand-by mode

As already said, to enter Stand-by Mode XRAM2EN bit in the XPERCON Register must be cleared (note that this bit is automatically reset by any kind of RESET event, see *Section 23: System reset on page 472*): this allows to freeze immediately the RAM interface, avoiding any data corruption. As a consequence of a RESET event, the RAM Power Supply is switched to the internal low-voltage supply V_{18SB} (derived from V_{STBY} through the low-power voltage regulator). The RAM interface will remain frozen until the bit XRAM2EN is set again by software initialization routine (at next exit from main V_{DD} Power-On reset sequence).

Since V_{18} is falling down (as a consequence of V_{DD} turning off), it can happen that the XRAM2EN bit is no longer able to guarantee its content (logic “0”), being the XPERCON Register powered by internal V_{18}. This does not generate any problem, because the Stand-by Mode switching dedicated circuit continues to confirm the RAM interface freezing, irrespective the XRAM2EN bit content; XRAM2EN bit status is considered again when internal V_{18} comes back over internal stand-by reference V_{18SB}.

If internal V_{18} becomes lower than internal stand-by reference (V_{18SB}) of about 0.3-0.45V with bit XRAM2EN set, the RAM Supply switching circuit is not active: in case of a temporary drop on internal V_{18} voltage versus internal V_{18SB} during normal code execution, no spurious Stand-by Mode switching can occur (the RAM is not frozen and can still be accessed).

The ST10F276 Core module, generating the RAM control signals, is powered by internal V_{18} supply; during turning off transient these control signals follow the V_{18}, while RAM is switched to V_{18SB} internal reference. It could happen that a high level of RAM write strobe

from ST10F276 Core (active low signal) is low enough to be recognized as a logic “0” by the RAM interface (due to V_{18} lower than V_{18SB}): the bus status could contain a valid address for the RAM and an unwanted data corruption could occur. For this reason, an extra interface, powered by the switched supply, is used to prevent the RAM from this kind of potential corruption mechanism.

WARNING:

During Power-Off phase, it is important that the external hardware maintains a stable ground level on $\overline{\text{RSTIN}}$ pin, without any glitch, in order to avoid spurious exiting from reset status with unstable power supply.

24.3.2 Exiting stand-by mode

After the system has entered the Stand-by Mode, the procedure to exit this mode consists of a standard Power-On sequence, with the only difference that the RAM is already powered through V_{18SB} internal reference (derived from V_{STBY} pin external voltage).

It is recommended to held the device under RESET ($\overline{\text{RSTIN}}$ pin forced low) until external V_{DD} voltage pin is stable. Even though, at the very beginning of the Power-On phase, the device is maintained under reset by the internal low voltage detector circuit (implemented inside the main voltage regulator) till the internal V_{18} becomes higher than about 1.0V, **there is no warranty that the device stays under reset status if RSTIN is at high level during power ramp up. So, it is important the external hardware is able to guarantee a stable ground level on RSTIN along the Power-On phase, without any temporary glitch.**

The external hardware should be responsible to drive low the $\overline{\text{RSTIN}}$ pin until the V_{DD} is stable, even though the internal LVD is active. Besides, it is requested an additional time (at least 1ms) to allow internal voltage regulator stabilization before releasing the $\overline{\text{RSTIN}}$ pin: this is necessary since the internal Flash has to begin its initialization phase (starting when $\overline{\text{RSTIN}}$ pin is released) with an already stable V_{18}.

Once the internal Reset signal goes low, the RAM (still frozen) power supply is switched to the main V_{18}.

At this time, everything becomes stable, and the execution of the initialization routines can start: XRAM2EN bit can be set, enabling the RAM.

24.3.3 Real time clock and stand-by mode

When Stand-by mode is entered (turning off the main supply V_{DD}), the Real Time Clock counting can be maintained running in case the on-chip low-power oscillator is used to provide the reference to the counter. This is not possible if the main oscillator is used as reference for the counter: being the main oscillator powered by V_{DD}, once this is switched off, the oscillator is stopped.

24.4 Output pin status

During idle mode the CPU clocks are turned off, while all peripherals continue their operation in the normal way. Therefore all ports pins, which are configured as general purpose output pins, output the last data value which was written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral. Port pins which are used for bus control functions go into that state which represents the inactive state of the respective function

($\overline{WR}$), or to a defined state which is based on the last bus access ($\overline{BHE}$). Port pins which are used as external address/data bus hold the address/data which was output during the last external memory access before entry into Idle mode under the following conditions:

P0H outputs the high byte of the last address if a multiplexed bus mode with 8-bit data bus is used, otherwise P0H is floating. P0L is always floating in Idle mode.

PORT1 outputs the lower 16 bits of the last address if a de-multiplexed bus mode is used, otherwise the output pins of PORT1 represent the port latch data.

Port4 outputs the segment address for the last access on those pins that were selected during reset, otherwise the output pins of Port4 represent the port latch data.

During power down mode the oscillator and the clocks to the CPU and to the peripherals are turned off. Like in Idle mode, all port pins which are configured as general purpose output pins output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

The *Table 67* summarizes the state of all ST10F276 output pins during Idle and Power Down mode.

Table 67. Output pin state during Idle and power down modes

ST10F276 output pin(s)	Idle mode		Power Down Mode	
	No external bus	External bus enabled	No external bus	External bus enabled
ALE	Low	Low	Low	Low
RD, WR	High	High	High	High
CLKOUT	Active	Active	High	High
RSTOUT	1	1	1	1
P0L	Port Latch Data	Floating	Port Latch Data	Floating
P0H	Port Latch Data	A15...A8 [2] / Float	Port Latch Data	A15...A8 [2] / Float
PORT1	Port Latch Data	Last Address [3] / Port Latch Data	Port Latch Data	Last Address [3] / Port Latch Data
Port4	Port Latch Data	Port Latch Data/Last segment	Port Latch Data	Port Latch Data/Last segment
BHE	Port Latch Data	Last value	Port Latch Data	Last value
HLDA	Port Latch Data	Last value	Port Latch Data	Last value
BREQ	Port Latch Data	High	Port Latch Data	High
CSx	Port Latch Data	Last value [4]	Port Latch Data	Last value [4]
Other Port Output Pins	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function

Note: 1 High if EINIT was executed before entering Idle or Power Down mode, Low otherwise.

2 For multiplexed buses with 8-bit data bus.

3 For de-multiplexed buses.

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

25 Programmable output clock divider

A specific register mapped on the XBUS allows to choose the division factor on the CLKOUT signal (P3.15). This register is mapped on X-Miscellaneous memory address range.

Real time clock and power down modeXBUS Reset Value: - - 00h

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

Bit	Function
DIV	**Clock Divider setting** '00h': $f_{CLKOUT} = f_{CPU}$ '01h': $f_{CLKOUT} = f_{CPU} / 2$ '02h': $f_{CLKOUT} = f_{CPU} / 3$ '03h': $f_{CLKOUT} = f_{CPU} / 4$ 'FFh': $f_{CLKOUT} = f_{CPU} / 256$

When CLKOUT function is enabled by setting bit CLKEN of register SYSCON, by default the CPU clock is output on P3.15. Setting bit XMISCEN of register XPERCON and bit XPEN of register SYSCON, it is possible to program the clock prescaling factor: in this way on P3.15 a prescaled value of the CPU clock can be output.

When CLKOUT function is not enabled (bit CLKEN of register SYSCON cleared), P3.15 does not output any clock signal, even though XCLKOUTDIV register is programmed.

26 Register set

This section summarizes all registers implemented in the ST10F276, and explains the description format used in the sections describing the function and layout of the SFRs. For easy reference the registers (except for GPRs) are ordered in two ways:

- Ordered by register name, to find the location of a specific register.
- Ordered by address, to check which register a given address references.

26.1 Register description format

Along the document, the function and the layout of the different registers is described in a specific format. The examples below explain this format.

A word register looks like this:

REG_NAME (A16h / A8h) SFR/ESFR/XBUS Reset Value: ****h

15	14	13	12	11	10	9	8	7	6	5 4 3	2 1 0
-	-	-	-	-	wr only	**hw bit**	rd only	std bit	**hw bit**	bit field	bit field
					W	RW	R	RW	RW	RW	RW

Bit	Function
bit(field) name	**Explanation of bit(field) name** Description of the functions controlled by this bit(field).

A byte register looks like this:

REG_NAME (A16h / A8h) SFR/ESFR/XBUS Reset Value: - - **h

15	14	13	12	11	10	9	8	7	6	5 4 3 2	1 0
-	-	-	-	-	-	-	-	std bit	**hw bit**	bit field	bit field
								RW	RW	RW	RW

Elements:

REG_NAME	Name of this register
A16h / A8h	Long 16-bit address / Short 8-bit address
SFR/ESFR/XBUS	Register space (SFR, ESFR or XBUS Register)
(* *) * *	Register contents after reset
	0/1: defined
	x: undefined (undefined ('**x**') after Power-On)
	U: unchanged
hw bit	bits that are set/cleared by hardware are written in **bold**
-	bits that are not implemented or reserved: never write to these bits

26.2 General purpose registers (GPRs)

The GPRs form the register bank that the CPU works with. This register bank may be located anywhere within the IRAM via the Context Pointer (CP). Due to the addressing mechanism, GPR banks can only reside within the IRAM. All GPRs are bit-addressable.

Table 68. 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 eight 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 69. 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

Table 69. General purpose registers (GPRs) bit wise addressing (continued)

Name	Physical address	8-bit address	Description	Reset value
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

26.3 Special function registers ordered by name

The following table lists all SFRs which are implemented in the ST10F276 in alphabetical order.
Bit-addressable SFRs are marked with the letter "**b**" in column "Name".
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter "**E**" in column "Physical Address".

Table 70. 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 Reg.	- - 00h
ADCON b	FFA0h	D0h	A/D Converter Control Register	0000h
ADDAT	FEA0h	50h	A/D Converter Result Register	0000h
ADDAT2	F0A0h E	50h	A/D Converter 2 Result Register	0000h
ADDRSEL1	FE18h	0Ch	Address Select Register 1	0000h
ADDRSEL2	FE1Ah	0Dh	Address Select Register 2	0000h
ADDRSEL3	FE1Ch	0Eh	Address Select Register 3	0000h
ADDRSEL4	FE1Eh	0Fh	Address Select Register 4	0000h
ADEIC b	FF9Ah	CDh	A/D Converter Overrun Error Interrupt Control Register	- - 00h
BUSCON0 b	FF0Ch	86h	Bus Configuration Register 0	0xx0h [1]
BUSCON1 b	FF14h	8Ah	Bus Configuration Register 1	0000h
BUSCON2 b	FF16h	8Bh	Bus Configuration Register 2	0000h
BUSCON3 b	FF18h	8Ch	Bus Configuration Register 3	0000h
BUSCON4 b	FF1Ah	8Dh	Bus Configuration Register 4	0000h
CAPREL	FE4Ah	25h	GPT2 Capture/Reload Register	0000h

Table 70. Special function registers ordered by name (continued)

Name	Physical address	8-bit address	Description	Reset value
CC0	FE80h	40h	CAPCOM Register 0	0000h
CC0IC b	FF78h	BCh	CAPCOM Register 0 Interrupt Control Register	- - 00h
CC1	FE82h	41h	CAPCOM Register 1	0000h
CC1IC b	FF7Ah	BDh	CAPCOM Register 1 Interrupt Control Register	- - 00h
CC2	FE84h	42h	CAPCOM Register 2	0000h
CC2IC b	FF7Ch	BEh	CAPCOM Register 2 Interrupt Control Register	- - 00h
CC3	FE86h	43h	CAPCOM Register 3	0000h
CC3IC b	FF7Eh	BFh	CAPCOM Register 3 Interrupt Control Register	- - 00h
CC4	FE88h	44h	CAPCOM Register 4	0000h
CC4IC b	FF80h	C0h	CAPCOM Register 4 Interrupt Control Register	- - 00h
CC5	FE8Ah	45h	CAPCOM Register 5	0000h
CC5IC b	FF82h	C1h	CAPCOM Register 5 Interrupt Control Register	- - 00h
CC6	FE8Ch	46h	CAPCOM Register 6	0000h
CC6IC b	FF84h	C2h	CAPCOM Register 6 Interrupt Control Register	- - 00h
CC7	FE8Eh	47h	CAPCOM Register 7	0000h
CC7IC b	FF86h	C3h	CAPCOM Register 7 Interrupt Control Register	- - 00h
CC8	FE90h	48h	CAPCOM Register 8	0000h
CC8IC b	FF88h	C4h	CAPCOM Register 8 Interrupt Control Register	- - 00h
CC9	FE92h	49h	CAPCOM Register 9	0000h
CC9IC b	FF8Ah	C5h	CAPCOM Register 9 Interrupt Control Register	- - 00h
CC10	FE94h	4Ah	CAPCOM Register 10	0000h
CC10IC b	FF8Ch	C6h	CAPCOM Register 10 Interrupt Control Register	- - 00h
CC11	FE96h	4Bh	CAPCOM Register 11	0000h
CC11IC b	FF8Eh	C7h	CAPCOM Register 11 Interrupt Control Register	- - 00h
CC12	FE98h	4Ch	CAPCOM Register 12	0000h
CC12IC b	FF90h	C8h	CAPCOM Register 12 Interrupt Control Register	- - 00h
CC13	FE9Ah	4Dh	CAPCOM Register 13	0000h
CC13IC b	FF92h	C9h	CAPCOM Register 13 Interrupt Control Register	- - 00h
CC14	FE9Ch	4Eh	CAPCOM Register 14	0000h
CC14IC b	FF94h	CAh	CAPCOM Register 14 Interrupt Control Register	- - 00h
CC15	FE9Eh	4Fh	CAPCOM Register 15	0000h
CC15IC b	FF96h	CBh	CAPCOM Register 15 Interrupt Control Register	- - 00h
CC16	FE60h	30h	CAPCOM Register 16	0000h
CC16IC b	F160h E	B0h	CAPCOM Register 16 Interrupt Control Register	- - 00h

Table 70. Special function registers ordered by name (continued)

Name	Physical address	8-bit address	Description	Reset value
CC17	FE62h	31h	CAPCOM Register 17	0000h
CC17IC b	F162h E	B1h	CAPCOM Register 17 Interrupt Control Register	- - 00h
CC18	FE64h	32h	CAPCOM Register 18	0000h
CC18IC b	F164h E	B2h	CAPCOM Register 18 Interrupt Control Register	- - 00h
CC19	FE66h	33h	CAPCOM Register 19	0000h
CC19IC b	F166h E	B3h	CAPCOM Register 19 Interrupt Control Register	- - 00h
CC20	FE68h	34h	CAPCOM Register 20	0000h
CC20IC b	F168h E	B4h	CAPCOM Register 20 Interrupt Control Register	- - 00h
CC21	FE6Ah	35h	CAPCOM Register 21	0000h
CC21IC b	F16Ah E	B5h	CAPCOM Register 21 Interrupt Control Register	- - 00h
CC22	FE6Ch	36h	CAPCOM Register 22	0000h
CC22IC b	F16Ch E	B6h	CAPCOM Register 22 Interrupt Control Register	- - 00h
CC23	FE6Eh	37h	CAPCOM Register 23	0000h
CC23IC b	F16Eh E	B7h	CAPCOM Register 23 Interrupt Control Register	- - 00h
CC24	FE70h	38h	CAPCOM Register 24	0000h
CC24IC b	F170h E	B8h	CAPCOM Register 24 Interrupt Control Register	- - 00h
CC25	FE72h	39h	CAPCOM Register 25	0000h
CC25IC b	F172h E	B9h	CAPCOM Register 25 Interrupt Control Register	- - 00h
CC26	FE74h	3Ah	CAPCOM Register 26	0000h
CC26IC b	F174h E	BAh	CAPCOM Register 26 Interrupt Control Register	- - 00h
CC27	FE76h	3Bh	CAPCOM Register 27	0000h
CC27IC b	F176h E	BBh	CAPCOM Register 27 Interrupt Control Register	- - 00h
CC28	FE78h	3Ch	CAPCOM Register 28	0000h
CC28IC b	F178h E	BCh	CAPCOM Register 28 Interrupt Control Register	- - 00h
CC29	FE7Ah	3Dh	CAPCOM Register 29	0000h
CC29IC b	F184h E	C2h	CAPCOM Register 29 Interrupt Control Register	- - 00h
CC30	FE7Ch	3Eh	CAPCOM Register 30	0000h
CC30IC b	F18Ch E	C6h	CAPCOM Register 30 Interrupt Control Register	- - 00h
CC31	FE7Eh	3Fh	CAPCOM Register 31	0000h
CC31IC b	F194h E	CAh	CAPCOM Register 31 Interrupt Control Register	- - 00h
CCM0 b	FF52h	A9h	CAPCOM Mode Control Register 0	0000h
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

Table 70. Special function registers ordered by name (continued)

Name	Physical address	8-bit address	Description	Reset value
CCM4 b	FF22h	91h	CAPCOM Mode Control Register 4	0000h
CCM5 b	FF24h	92h	CAPCOM Mode Control Register 5	0000h
CCM6 b	FF26h	93h	CAPCOM Mode Control Register 6	0000h
CCM7 b	FF28h	94h	CAPCOM Mode Control Register 7	0000h
CP	FE10h	08h	CPU Context Pointer Register	FC00h
CRIC b	FF6Ah	B5h	GPT2 CAPREL Interrupt Control Register	- - 00h
CSP	FE08h	04h	CPU Code Segment Pointer Register (read only)	0000h
DP0L b	F100h E	80h	P0L Direction Control Register	- - 00h
DP0H b	F102h E	81h	P0h Direction Control Register	- - 00h
DP1L b	F104h E	82h	P1L Direction Control Register	- - 00h
DP1H b	F106h E	83h	P1h Direction Control Register	- - 00h
DP2 b	FFC2h	E1h	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
EMUCON	FE0Ah	05h	Emulation Control Register	- - xxh
EXICON b	F1C0h E	E0h	External Interrupt Control Register	0000h
EXISEL b	F1DAh E	EDh	External Interrupt Source Selection Register	0000h
IDCHIP	F07Ch E	3Eh	Device Identifier Register (n is the device revision)	114nh
IDMANUF	F07Eh E	3Fh	Manufacturer Identifier Register	0403h
IDMEM	F07Ah E	3Dh	On-chip Memory Identifier Register	30D0h
IDPROG	F078h E	3Ch	Programming Voltage Identifier Register	0040h
IDX0 b	FF08h	84h	MAC Unit Address Pointer 0	0000h
IDX1 b	FF0Ah	85h	MAC Unit Address Pointer 1	0000h
MAH	FE5Eh	2Fh	MAC Unit Accumulator - High Word	0000h
MAL	FE5Ch	2Eh	MAC Unit Accumulator - Low Word	0000h
MCW b	FFDCh	EEh	MAC Unit Control Word	0000h
MDC b	FF0Eh	87h	CPU Multiply Divide Control Register	0000h

Table 70. Special function registers ordered by name (continued)

Name	Physical address	8-bit address	Description	Reset value
MDH	FE0Ch	06h	CPU Multiply Divide Register – High Word	0000h
MDL	FE0Eh	07h	CPU Multiply Divide Register – Low Word	0000h
MRW b	FFDAh	EDh	MAC Unit Repeat Word	0000h
MSW b	FFDEh	EFh	MAC Unit Status Word	0200h
ODP2 b	F1C2h E	E1h	Port2 Open Drain Control Register	0000h
ODP3 b	F1C6h E	E3h	Port3 Open Drain Control Register	0000h
ODP4 b	F1CAh E	E5h	Port4 Open Drain Control Register	- - 00h
ODP6 b	F1CEh E	E7h	Port6 Open Drain Control Register	- - 00h
ODP7 b	F1D2h E	E9h	Port7 Open Drain Control Register	- - 00h
ODP8 b	F1D6h E	EBh	Port8 Open Drain Control Register	- - 00h
ONES b	FF1Eh	8Fh	Constant Value 1's Register (read only)	FFFFh
P0L b	FF00h	80h	PORT0 Low Register (Lower half of PORT0)	- - 00h
P0H b	FF02h	81h	PORT0 High Register (Upper half of PORT0)	- - 00h
P1L b	FF04h	82h	PORT1 Low Register (Lower half of PORT1)	- - 00h
P1H b	FF06h	83h	PORT1 High Register (Upper half of PORT1)	- - 00h
P2b	FFC0h	E0h	Port2 Register	0000h
P3b	FFC4h	E2h	Port3 Register	0000h
P4b	FFC8h	E4h	Port4 Register (8-bit)	- - 00h
P5b	FFA2h	D1h	Port5 Register (read only)	xxxxh
P6b	FFCCh	E6h	Port6 Register (8-bit)	- - 00h
P7b	FFD0h	E8h	Port7 Register (8-bit)	- - 00h
P8b	FFD4h	EAh	Port8 Register (8-bit)	- - 00h
P5DIDIS b	FFA4h	D2h	Port5 Digital Disable Register	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
PICON b	F1C4h E	E2h	Port Input Threshold Control Register	- - 00h
PP0	F038h E	1Ch	PWM Module Period Register 0	0000h
PP1	F03Ah E	1Dh	PWM Module Period Register 1	0000h

Table 70. Special function registers ordered by name (continued)

Name	Physical address	8-bit address	Description	Reset value
PP2	F03Ch E	1Eh	PWM Module Period Register 2	0000h
PP3	F03Eh E	1Fh	PWM Module Period Register 3	0000h
PSW b	FF10h	88h	CPU Program Status Word	0000h
PT0	F030h E	18h	PWM Module Up/Down Counter 0	0000h
PT1	F032h E	19h	PWM Module Up/Down Counter 1	0000h
PT2	F034h E	1Ah	PWM Module Up/Down Counter 2	0000h
PT3	F036h E	1Bh	PWM Module Up/Down Counter 3	0000h
PW0	FE30h	18h	PWM Module Pulse Width Register 0	0000h
PW1	FE32h	19h	PWM Module Pulse Width Register 1	0000h
PW2	FE34h	1Ah	PWM Module Pulse Width Register 2	0000h
PW3	FE36h	1Bh	PWM Module Pulse Width Register 3	0000h
PWMCON0 b	FF30h	98h	PWM Module Control Register 0	0000h
PWMCON1 b	FF32h	99h	PWM Module Control Register 1	0000h
PWMIC b	F17Eh E	BFh	PWM Module Interrupt Control Register	- - 00h
QR0	F004h E	02h	MAC Unit Offset Register R0	0000h
QR1	F006h E	03h	MAC Unit Offset Register R1	0000h
QX0	F000h E	00h	MAC Unit Offset Register X0	0000h
QX1	F002h E	01h	MAC Unit Offset Register X1	0000h
RP0H b	F108h E	84h	System Start-up Configuration Register (read only)	- - xxh
S0BG	FEB4h	5Ah	Serial Channel 0 Baud Rate Generator Reload Register	0000h
S0CON b	FFB0h	D8h	Serial Channel 0 Control Register	0000h
S0EIC b	FF70h	B8h	Serial Channel 0 Error Interrupt Control Register	- - 00h
S0RBUF	FEB2h	59h	Serial Channel 0 Receive Buffer Register (read only)	- xxxh
S0RIC b	FF6Eh	B7h	Serial Channel 0 Receive Interrupt Control Register	- - 00h
S0TBIC b	F19Ch E	CEh	Serial Channel 0 Transmit Buffer Interrupt Control Reg.	- - 00h
S0TBUF	FEB0h	58h	Serial Channel 0 Transmit Buffer Register (write only)	0000h
S0TIC b	FF6Ch	B6h	Serial Channel 0 Transmit Interrupt Control Register	- - 00h
SP	FE12h	09h	CPU System Stack Pointer Register	FC00h
SSCBR	F0B4h E	5Ah	SSC Baud Rate Register	0000h
SSCCON b	FFB2h	D9h	SSC Control Register	0000h
SSCEIC b	FF76h	BBh	SSC Error Interrupt Control Register	- - 00h
SSCRB	F0B2h E	59h	SSC Receive Buffer (read only)	xxxxh
SSCRIC b	FF74h	BAh	SSC Receive Interrupt Control Register	- - 00h
SSCTB	F0B0h E	58h	SSC Transmit Buffer (write only)	0000h

Table 70. Special function registers ordered by name (continued)

Name	Physical address	8-bit address	Description	Reset value
SSCTIC b	FF72h	B9h	SSC Transmit Interrupt Control Register	- - 00h
STKOV	FE14h	0Ah	CPU Stack Overflow Pointer Register	FA00h
STKUN	FE16h	0Bh	CPU Stack Underflow Pointer Register	FC00h
SYSCON b	FF12h	89h	CPU System Configuration Register	0xx0h [2]
T0	FE50h	28h	CAPCOM Timer 0 Register	0000h
T01CON b	FF50h	A8h	CAPCOM Timer 0 and Timer 1 Control Register	0000h
T0IC b	FF9Ch	CEh	CAPCOM Timer 0 Interrupt Control Register	- - 00h
T0REL	FE54h	2Ah	CAPCOM Timer 0 Reload Register	0000h
T1	FE52h	29h	CAPCOM Timer 1 Register	0000h
T1IC b	FF9Eh	CFh	CAPCOM Timer 1 Interrupt Control Register	- - 00h
T1REL	FE56h	2Bh	CAPCOM Timer 1 Reload Register	0000h
T2	FE40h	20h	GPT1 Timer 2 Register	0000h
T2CON b	FF40h	A0h	GPT1 Timer 2 Control Register	0000h
T2IC b	FF60h	B0h	GPT1 Timer 2 Interrupt Control Register	- - 00h
T3	FE42h	21h	GPT1 Timer 3 Register	0000h
T3CON b	FF42h	A1h	GPT1 Timer 3 Control Register	0000h
T3IC b	FF62h	B1h	GPT1 Timer 3 Interrupt Control Register	- - 00h
T4	FE44h	22h	GPT1 Timer 4 Register	0000h
T4CON b	FF44h	A2h	GPT1 Timer 4 Control Register	0000h
T4IC b	FF64h	B2h	GPT1 Timer 4 Interrupt Control Register	- - 00h
T5	FE46h	23h	GPT2 Timer 5 Register	0000h
T5CON b	FF46h	A3h	GPT2 Timer 5 Control Register	0000h
T5IC b	FF66h	B3h	GPT2 Timer 5 Interrupt Control Register	- - 00h
T6	FE48h	24h	GPT2 Timer 6 Register	0000h
T6CON b	FF48h	A4h	GPT2 Timer 6 Control Register	0000h
T6IC b	FF68h	B4h	GPT2 Timer 6 Interrupt Control Register	- - 00h
T7	F050h E	28h	CAPCOM Timer 7 Register	0000h
T78CON b	FF20h	90h	CAPCOM Timer 7 and 8 Control Register	0000h
T7IC b	F17Ah E	BDh	CAPCOM Timer 7 Interrupt Control Register	- - 00h
T7REL	F054h E	2Ah	CAPCOM Timer 7 Reload Register	0000h
T8	F052h E	29h	CAPCOM Timer 8 Register	0000h
T8IC b	F17Ch E	BEh	CAPCOM Timer 8 Interrupt Control Register	- - 00h
T8REL	F056h E	2Bh	CAPCOM Timer 8 Reload Register	0000h
TFRb	FFACh	D6h	Trap Flag Register	0000h

Table 70. Special function registers ordered by name (continued)

Name	Physical address	8-bit address	Description	Reset value
WDT	FEAEh	57h	Watchdog Timer Register (read only)	0000h
WDTCON b	FFAEh	D7h	Watchdog Timer Control Register	00xxh [3]
XADRS3	F01Ch E	0Eh	XPER Address Select Register 3	800Bh
XP0IC b	F186h E	C3h	See *Section 5.7: X-peripheral interrupt*	- - 00h [4]
XP1IC b	F18Eh E	C7h	See *Section 5.7: X-peripheral interrupt*	- - 00h [4]
XP2IC b	F196h E	CBh	See *Section 5.7: X-peripheral interrupt*	- - 00h [4]
XP3IC b	F19Eh E	CFh	See *Section 5.7: X-peripheral interrupt*	- - 00h [4]
XPERCON	F024h E	12h	XPER Configuration Register	- - 05h
ZEROS b	FF1Ch	8Eh	Constant Value 0's Register (read only)	0000h

Note: 1 The BUSCON0 reset value is: 0000 0xx0 xx00 0000b.

2 The system configuration is selected during reset: in particular, the SYSCON reset value is 0000 0xx0 x000 0000b.

3 Reset Value depends on different triggered reset event.

4 The XPnIC Interrupt Control Registers control interrupt requests from integrated X-Bus peripherals. Some software controlled interrupt requests may be generated by setting the XPnIR bits (of XPnIC register) of the unused X-Peripheral nodes.

26.4 Special function registers ordered by address

The following table lists all SFRs which are implemented in the ST10F276 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 71. Special function registers ordered by address

Name	Physical address	8-bit address	Description	Reset value
QX0	F000h E	00h	MAC Unit Offset Register X0	0000h
QX1	F002h E	01h	MAC Unit Offset Register X1	0000h
QR0	F004h E	02h	MAC Unit Offset Register R0	0000h
QR1	F006h E	03h	MAC Unit Offset Register R1	0000h
XADRS3	F01Ch E	0Eh	XPER Address Select Register 3	800Bh
XPERCON	F024h E	12h	XPER Configuration Register	- - 05h
PT0	F030h E	18h	PWM Module Up/Down Counter 0	0000h
PT1	F032h E	19h	PWM Module Up/Down Counter 1	0000h
PT2	F034h E	1Ah	PWM Module Up/Down Counter 2	0000h

Table 71. Special function registers ordered by address (continued)

Name	Physical address	8-bit address	Description	Reset value
PT3	F036h E	1Bh	PWM Module Up/Down Counter 3	0000h
PP0	F038h E	1Ch	PWM Module Period Register 0	0000h
PP1	F03Ah E	1Dh	PWM Module Period Register 1	0000h
PP2	F03Ch E	1Eh	PWM Module Period Register 2	0000h
PP3	F03Eh E	1Fh	PWM Module Period Register 3	0000h
T7	F050h E	28h	CAPCOM Timer 7 Register	0000h
T8	F052h E	29h	CAPCOM Timer 8 Register	0000h
T7REL	F054h E	2Ah	CAPCOM Timer 7 Reload Register	0000h
T8REL	F056h E	2Bh	CAPCOM Timer 8 Reload Register	0000h
IDPROG	F078h E	3Ch	Programming Voltage Identifier Register	0040h
IDMEM	F07Ah E	3Dh	On-chip Memory Identifier Register	30D0h
IDCHIP	F07Ch E	3Eh	Device Identifier Register (n is the device revision)	114nh
IDMANUF	F07Eh E	3Fh	Manufacturer Identifier Register	0403h
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
DP0Lb	F100h E	80h	P0L Direction Control Register	- - 00h
DP0Hb	F102h E	81h	P0H Direction Control Register	- - 00h
DP1Lb	F104h E	82h	P1L Direction Control Register	- - 00h
DP1Hb	F106h E	83h	P1H Direction Control Register	- - 00h
RP0Hb	F108h E	84h	System Start-up Configuration Register (Read only)	- - xxh
CC16ICb	F160h E	B0h	CAPCOM Register 16 Interrupt Control Register	- - 00h
CC17ICb	F162h E	B1h	CAPCOM Register 17 Interrupt Control Register	- - 00h
CC18ICb	F164h E	B2h	CAPCOM Register 18 Interrupt Control Register	- - 00h
CC19ICb	F166h E	B3h	CAPCOM Register 19 Interrupt Control Register	- - 00h
CC20ICb	F168h E	B4h	CAPCOM Register 20 Interrupt Control Register	- - 00h
CC21ICb	F16Ah E	B5h	CAPCOM Register 21 Interrupt Control Register	- - 00h
CC22ICb	F16Ch E	B6h	CAPCOM Register 22 Interrupt Control Register	- - 00h
CC23ICb	F16Eh E	B7h	CAPCOM Register 23 Interrupt Control Register	- - 00h
CC24ICb	F170h E	B8h	CAPCOM Register 24 Interrupt Control Register	- - 00h
CC25ICb	F172h E	B9h	CAPCOM Register 25 Interrupt Control Register	- - 00h
CC26ICb	F174h E	BAh	CAPCOM Register 26 Interrupt Control Register	- - 00h
CC27ICb	F176h E	BBh	CAPCOM Register 27 Interrupt Control Register	- - 00h

Table 71. Special function registers ordered by address (continued)

Name	Physical address	8-bit address	Description	Reset value
CC28IC b	F178h E	BCh	CAPCOM Register 28 Interrupt Control Register	- - 00h
T7IC b	F17Ah E	BDh	CAPCOM Timer 7 Interrupt Control Register	- - 00h
T8IC b	F17Ch E	BEh	CAPCOM Timer 8 Interrupt Control Register	- - 00h
PWMIC b	F17Eh E	BFh	PWM Module Interrupt Control Register	- - 00h
CC29IC b	F184h E	C2h	CAPCOM Register 29 Interrupt Control Register	- - 00h
XP0IC b	F186h E	C3h	See *Section 5.7: X-peripheral interrupt*	- - 00h
CC30IC b	F18Ch E	C6h	CAPCOM Register 30 Interrupt Control Register	- - 00h
XP1IC b	F18Eh E	C7h	See *Section 5.7: X-peripheral interrupt*	- - 00h
CC31IC b	F194h E	CAh	CAPCOM Register 31 Interrupt Control Register	- - 00h
XP2IC b	F196h E	CBh	See *Section 5.7: X-peripheral interrupt*	- - 00h
S0TBIC b	F19Ch E	CEh	Serial Channel 0 Transmit Buffer Interrupt Control Reg.	- - 00h
XP3IC b	F19Eh E	CFh	See *Section 5.7: X-peripheral interrupt*	- - 00h
EXICON b	F1C0h E	E0h	External Interrupt Control Register	0000h
ODP2 b	F1C2h E	E1h	Port2 Open Drain Control Register	0000h
PICON b	F1C4h E	E2h	Port Input Threshold Control Register	- - 00h
ODP3 b	F1C6h E	E3h	Port3 Open Drain Control Register	0000h
ODP4 b	F1CAh E	E5h	Port4 Open Drain Control Register	- - 00h
ODP6 b	F1CEh E	E7h	Port6 Open Drain Control Register	- - 00h
ODP7 b	F1D2h E	E9h	Port7 Open Drain Control Register	- - 00h
ODP8 b	F1D6h E	EBh	Port8 Open Drain Control Register	- - 00h
EXISEL b	F1DAh E	EDh	External Interrupt Source Selection Register	0000h
DPP0	FE00h	00h	CPU Data Page Pointer 0 Register (10-bit)	0000h
DPP1	FE02h	01h	CPU Data Page Pointer 1 Register (10-bit)	0001h
DPP2	FE04h	02h	CPU Data Page Pointer 2 Register (10-bit)	0002h
DPP3	FE06h	03h	CPU Data Page Pointer 3 Register (10-bit)	0003h
CSP	FE08h	04h	CPU Code Segment Pointer Register (read only)	0000h
EMUCON	FE0Ah	05h	Emulation Control Register	- - xxh
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

Table 71. Special function registers ordered by address (continued)

Name	Physical address	8-bit address	Description	Reset value
ADDRSEL2	FE1Ah	0Dh	Address Select Register 2	0000h
ADDRSEL3	FE1Ch	0Eh	Address Select Register 3	0000h
ADDRSEL4	FE1Eh	0Fh	Address Select Register 4	0000h
PW0	FE30h	18h	PWM Module Pulse Width Register 0	0000h
PW1	FE32h	19h	PWM Module Pulse Width Register 1	0000h
PW2	FE34h	1Ah	PWM Module Pulse Width Register 2	0000h
PW3	FE36h	1Bh	PWM Module Pulse Width Register 3	0000h
T2	FE40h	20h	GPT1 Timer 2 Register	0000h
T3	FE42h	21h	GPT1 Timer 3 Register	0000h
T4	FE44h	22h	GPT1 Timer 4 Register	0000h
T5	FE46h	23h	GPT2 Timer 5 Register	0000h
T6	FE48h	24h	GPT2 Timer 6 Register	0000h
CAPREL	FE4Ah	25h	GPT2 Capture/Reload Register	0000h
T0	FE50h	28h	CAPCOM Timer 0 Register	0000h
T1	FE52h	29h	CAPCOM Timer 1 Register	0000h
T0REL	FE54h	2Ah	CAPCOM Timer 0 Reload Register	0000h
T1REL	FE56h	2Bh	CAPCOM Timer 1 Reload Register	0000h
MAL	FE5Ch	2Eh	MAC Unit Accumulator - Low Word	0000h
MAH	FE5Eh	2Fh	MAC Unit Accumulator - High Word	0000h
CC16	FE60h	30h	CAPCOM Register 16	0000h
CC17	FE62h	31h	CAPCOM Register 17	0000h
CC18	FE64h	32h	CAPCOM Register 18	0000h
CC19	FE66h	33h	CAPCOM Register 19	0000h
CC20	FE68h	34h	CAPCOM Register 20	0000h
CC21	FE6Ah	35h	CAPCOM Register 21	0000h
CC22	FE6Ch	36h	CAPCOM Register 22	0000h
CC23	FE6Eh	37h	CAPCOM Register 23	0000h
CC24	FE70h	38h	CAPCOM Register 24	0000h
CC25	FE72h	39h	CAPCOM Register 25	0000h
CC26	FE74h	3Ah	CAPCOM Register 26	0000h
CC27	FE76h	3Bh	CAPCOM Register 27	0000h
CC28	FE78h	3Ch	CAPCOM Register 28	0000h
CC29	FE7Ah	3Dh	CAPCOM Register 29	0000h
CC30	FE7Ch	3Eh	CAPCOM Register 30	0000h

Table 71. Special function registers ordered by address (continued)

Name	Physical address	8-bit address	Description	Reset value
CC31	FE7Eh	3Fh	CAPCOM Register 31	0000h
CC0	FE80h	40h	CAPCOM Register 0	0000h
CC1	FE82h	41h	CAPCOM Register 1	0000h
CC2	FE84h	42h	CAPCOM Register 2	0000h
CC3	FE86h	43h	CAPCOM Register 3	0000h
CC4	FE88h	44h	CAPCOM Register 4	0000h
CC5	FE8Ah	45h	CAPCOM Register 5	0000h
CC6	FE8Ch	46h	CAPCOM Register 6	0000h
CC7	FE8Eh	47h	CAPCOM Register 7	0000h
CC8	FE90h	48h	CAPCOM Register 8	0000h
CC9	FE92h	49h	CAPCOM Register 9	0000h
CC10	FE94h	4Ah	CAPCOM Register 10	0000h
CC11	FE96h	4Bh	CAPCOM Register 11	0000h
CC12	FE98h	4Ch	CAPCOM Register 12	0000h
CC13	FE9Ah	4Dh	CAPCOM Register 13	0000h
CC14	FE9Ch	4Eh	CAPCOM Register 14	0000h
CC15	FE9Eh	4Fh	CAPCOM Register 15	0000h
ADDAT	FEA0h	50h	A/D Converter Result Register	0000h
WDT	FEAEh	57h	Watchdog Timer Register (read only)	0000h
S0TBUF	FEB0h	58h	Serial Channel 0 Transmit Buffer Register (write only)	0000h
S0RBUF	FEB2h	59h	Serial Channel 0 Receive Buffer Register (read only)	- xxxh
S0BG	FEB4h	5Ah	Serial Channel 0 Baud Rate Generator Reload Register	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

Table 71. Special function registers ordered by address (continued)

Name	Physical address	8-bit address	Description	Reset value
IDX0 b	FF08h	84h	MAC Unit Address Pointer 0	0000h
IDX1 b	FF0Ah	85h	MAC Unit Address Pointer 1	0000h
BUSCON0 b	FF0Ch	86h	Bus Configuration Register 0	0xx0h
MDC b	FF0Eh	87h	CPU Multiply Divide Control Register	0000h
PSW b	FF10h	88h	CPU Program Status word	0000h
SYSCON b	FF12h	89h	CPU System Configuration Register	0xx0h
BUSCON1 b	FF14h	8Ah	Bus Configuration Register 1	0000h
BUSCON2 b	FF16h	8Bh	Bus Configuration Register 2	0000h
BUSCON3 b	FF18h	8Ch	Bus Configuration Register 3	0000h
BUSCON4 b	FF1Ah	8Dh	Bus Configuration Register 4	0000h
ZEROS b	FF1Ch	8Eh	Constant Value 0's Register (read only)	0000h
ONES b	FF1Eh	8Fh	Constant Value 1's Register (read only)	FFFFh
T78CONb	FF20h	90h	CAPCOM Timer 7 and 8 Control Register	0000h
CCM4 b	FF22h	91h	CAPCOM Mode Control Register 4	0000h
CCM5 b	FF24h	92h	CAPCOM Mode Control Register 5	0000h
CCM6 b	FF26h	93h	CAPCOM Mode Control Register 6	0000h
CCM7 b	FF28h	94h	CAPCOM Mode Control Register 7	0000h
PWMCON0 b	FF30h	98h	PWM Module Control Register 0	0000h
PWMCON1 b	FF32h	99h	PWM Module Control Register 1	0000h
T2CON b	FF40h	A0h	GPT1 Timer 2 Control Register	0000h
T3CON b	FF42h	A1h	GPT1 Timer 3 Control Register	0000h
T4CON b	FF44h	A2h	GPT1 Timer 4 Control Register	0000h
T5CON b	FF46h	A3h	GPT2 Timer 5 Control Register	0000h
T6CON b	FF48h	A4h	GPT2 Timer 6 Control Register	0000h
T01CON b	FF50h	A8h	CAPCOM Timer 0 and Timer 1 Control Register	0000h
CCM0 b	FF52h	A9h	CAPCOM Mode Control Register 0	0000h
CCM1 b	FF54h	AAh	CAPCOM Mode Control Register 1	0000h
CCM2 b	FF56h	ABh	CAPCOM Mode Control Register 2	0000h
CCM3 b	FF58h	ACh	CAPCOM Mode Control Register 3	0000h
T2IC b	FF60h	B0h	GPT1 Timer 2 Interrupt Control Register	- - 00h
T3IC b	FF62h	B1h	GPT1 Timer 3 Interrupt Control Register	- - 00h
T4IC b	FF64h	B2h	GPT1 Timer 4 Interrupt Control Register	- - 00h
T5IC b	FF66h	B3h	GPT2 Timer 5 Interrupt Control Register	- - 00h
T6IC b	FF68h	B4h	GPT2 Timer 6 Interrupt Control Register	- - 00h

Table 71. Special function registers ordered by address (continued)

Name	Physical address	8-bit address	Description	Reset value
CRIC b	FF6Ah	B5h	GPT2 CAPREL Interrupt Control Register	- - 00h
S0TIC b	FF6Ch	B6h	Serial Channel 0 Transmit Interrupt Control Register	- - 00h
S0RIC b	FF6Eh	B7h	Serial Channel 0 Receive Interrupt Control Register	- - 00h
S0EIC b	FF70h	B8h	Serial Channel 0 Error Interrupt Control Register	- - 00h
SSCTIC b	FF72h	B9h	SSC Transmit Interrupt Control Register	- - 00h
SSCRIC b	FF74h	BAh	SSC Receive Interrupt Control Register	- - 00h
SSCEIC b	FF76h	BBh	SSC Error Interrupt Control Register	- - 00h
CC0IC b	FF78h	BCh	CAPCOM Register 0 Interrupt Control Register	- - 00h
CC1IC b	FF7Ah	BDh	CAPCOM Register 1 Interrupt Control Register	- - 00h
CC2IC b	FF7Ch	BEh	CAPCOM Register 2 Interrupt Control Register	- - 00h
CC3IC b	FF7Eh	BFh	CAPCOM Register 3 Interrupt Control Register	- - 00h
CC4IC b	FF80h	C0h	CAPCOM Register 4 Interrupt Control Register	- - 00h
CC5IC b	FF82h	C1h	CAPCOM Register 5 Interrupt Control Register	- - 00h
CC6IC b	FF84h	C2h	CAPCOM Register 6 Interrupt Control Register	- - 00h
CC7IC b	FF86h	C3h	CAPCOM Register 7 Interrupt Control Register	- - 00h
CC8IC b	FF88h	C4h	CAPCOM Register 8 Interrupt Control Register	- - 00h
CC9IC b	FF8Ah	C5h	CAPCOM Register 9 Interrupt Control Register	- - 00h
CC10IC b	FF8Ch	C6h	CAPCOM Register 10 Interrupt Control Register	- - 00h
CC11IC b	FF8Eh	C7h	CAPCOM Register 11 Interrupt Control Register	- - 00h
CC12IC b	FF90h	C8h	CAPCOM Register 12 Interrupt Control Register	- - 00h
CC13IC b	FF92h	C9h	CAPCOM Register 13 Interrupt Control Register	- - 00h
CC14IC b	FF94h	CAh	CAPCOM Register 14 Interrupt Control Register	- - 00h
CC15IC b	FF96h	CBh	CAPCOM Register 15 Interrupt Control Register	- - 00h
ADCIC b	FF98h	CCh	A/D Converter End of Conversion Interrupt Control Reg.	- - 00h
ADEIC b	FF9Ah	CDh	A/D Converter Overrun Error Interrupt Control Reg	- - 00h
T0IC b	FF9Ch	CEh	CAPCOM Timer 0 Interrupt Control Register	- - 00h
T1IC b	FF9Eh	CFh	CAPCOM Timer 1 Interrupt Control Register	- - 00h
ADCON b	FFA0h	D0h	A/D Converter Control Register	0000h
P5b	FFA2h	D1h	Port5 Register (read only)	xxxxh
P5DIDIS b	FFA4h	D2h	Port5 Digital Disable Register	0000h
TFR b	FFACh	D6h	Trap Flag Register	0000h
WDTCON b	FFAEh	D7h	Watchdog Timer Control Register	00xxh
S0CON b	FFB0h	D8h	Serial Channel 0 Control Register	0000h
SSCCON b	FFB2h	D9h	SSC Control Register	0000h

Table 71. Special function registers ordered by address (continued)

Name	Physical address	8-bit address	Description	Reset value
P2b	FFC0h	E0h	Port2 Register	0000h
DP2b	FFC2h	E1h	Port2 Direction Control Register	0000h
P3b	FFC4h	E2h	Port3 Register	0000h
DP3b	FFC6h	E3h	Port3 Direction Control Register	0000h
P4b	FFC8h	E4h	Port4 Register (8-bit)	- - 00h
DP4b	FFCAh	E5h	Port4 Direction Control Register	- - 00h
P6b	FFCCh	E6h	Port6 Register (8-bit)	- - 00h
DP6b	FFCEh	E7h	Port6 Direction Control Register	- - 00h
P7b	FFD0h	E8h	Port7 Register (8-bit)	- - 00h
DP7b	FFD2h	E9h	Port7 Direction Control Register	- - 00h
P8b	FFD4h	EAh	Port8 Register (8-bit)	- - 00h
DP8b	FFD6h	EBh	Port8 Direction Control Register	- - 00h
MRW b	FFDAh	EDh	MAC Unit Repeat Word	0000h
MCW b	FFDCh	EEh	MAC Unit Control Word	0000h
MSW b	FFDEh	EFh	MAC Unit Status Word	0200h

26.5 X-Registers ordered by name

The following table lists all X-Bus registers which are implemented in the ST10F276 ordered by their name. The Flash control registers are listed in a separate section, in spite of they are also physically mapped on X-Bus memory space. Note that all X-Registers are not bit-addressable.

Table 72. X-Registers ordered by name

Name	Physical address	Description	Reset value
CAN1BRPER	EF0Ch	CAN1: BRP Extension Register	0000h
CAN1BTR	EF06h	CAN1: Bit Timing Register	2301h
CAN1CR	EF00h	CAN1: CAN Control Register	0001h
CAN1EC	EF04h	CAN1: Error Counter	0000h
CAN1IF1A1	EF18h	CAN1: IF1 Arbitration 1	0000h
CAN1IF1A2	EF1Ah	CAN1: IF1 Arbitration 2	0000h
CAN1IF1CM	EF12h	CAN1: IF1 Command Mask	0000h
CAN1IF1CR	EF10h	CAN1: IF1 Command Request	0001h
CAN1IF1DA1	EF1Eh	CAN1: IF1 Data A 1	0000h
CAN1IF1DA2	EF20h	CAN1: IF1 Data A 2	0000h

Table 72. X-Registers ordered by name (continued)

Name	Physical address	Description	Reset value
CAN1IF1DB1	EF22h	CAN1: IF1 Data B 1	0000h
CAN1IF1DB2	EF24h	CAN1: IF1 Data B 2	0000h
CAN1IF1M1	EF14h	CAN1: IF1 Mask 1	FFFFh
CAN1IF1M2	EF16h	CAN1: IF1 Mask 2	FFFFh
CAN1IF1MC	EF1Ch	CAN1: IF1 Message Control	0000h
CAN1IF2A1	EF48h	CAN1: IF2 Arbitration 1	0000h
CAN1IF2A2	EF4Ah	CAN1: IF2 Arbitration 2	0000h
CAN1IF2CM	EF42h	CAN1: IF2 Command Mask	0000h
CAN1IF2CR	EF40h	CAN1: IF2 Command Request	0001h
CAN1IF2DA1	EF4Eh	CAN1: IF2 Data A 1	0000h
CAN1IF2DA2	EF50h	CAN1: IF2 Data A 2	0000h
CAN1IF2DB1	EF52h	CAN1: IF2 Data B 1	0000h
CAN1IF2DB2	EF54h	CAN1: IF2 Data B 2	0000h
CAN1IF2M1	EF44h	CAN1: IF2 Mask 1	FFFFh
CAN1IF2M2	EF46h	CAN1: IF2 Mask 2	FFFFh
CAN1IF2MC	EF4Ch	CAN1: IF2 Message Control	0000h
CAN1IP1	EFA0h	CAN1: Interrupt Pending 1	0000h
CAN1IP2	EFA2h	CAN1: Interrupt Pending 2	0000h
CAN1IR	EF08h	CAN1: Interrupt Register	0000h
CAN1MV1	EFB0h	CAN1: Message Valid 1	0000h
CAN1MV2	EFB2h	CAN1: Message Valid 2	0000h
CAN1ND1	EF90h	CAN1: New Data 1	0000h
CAN1ND2	EF92h	CAN1: New Data 2	0000h
CAN1SR	EF02h	CAN1: Status Register	0000h
CAN1TR	EF0Ah	CAN1: Test Register	00x0h
CAN1TR1	EF80h	CAN1: Transmission Request 1	0000h
CAN1TR2	EF82h	CAN1: Transmission Request 2	0000h
CAN2BRPER	EE0Ch	CAN2: BRP Extension Register	0000h
CAN2BTR	EE06h	CAN2: Bit Timing Register	2301h
CAN2CR	EE00h	CAN2: CAN Control Register	0001h
CAN2EC	EE04h	CAN2: Error Counter	0000h
CAN2IF1A1	EE18h	CAN2: IF1 Arbitration 1	0000h
CAN2IF1A2	EE1Ah	CAN2: IF1 Arbitration 2	0000h
CAN2IF1CM	EE12h	CAN2: IF1 Command Mask	0000h

Table 72. X-Registers ordered by name (continued)

Name	Physical address	Description	Reset value
CAN2IF1CR	EE10h	CAN2: IF1 Command Request	0001h
CAN2IF1DA1	EE1Eh	CAN2: IF1 Data A 1	0000h
CAN2IF1DA2	EE20h	CAN2: IF1 Data A 2	0000h
CAN2IF1DB1	EE22h	CAN2: IF1 Data B 1	0000h
CAN2IF1DB2	EE24h	CAN2: IF1 Data B 2	0000h
CAN2IF1M1	EE14h	CAN2: IF1 Mask 1	FFFFh
CAN2IF1M2	EE16h	CAN2: IF1 Mask 2	FFFFh
CAN2IF1MC	EE1Ch	CAN2: IF1 Message Control	0000h
CAN2IF2A1	EE48h	CAN2: IF2 Arbitration 1	0000h
CAN2IF2A2	EE4Ah	CAN2: IF2 Arbitration 2	0000h
CAN2IF2CM	EE42h	CAN2: IF2 Command Mask	0000h
CAN2IF2CR	EE40h	CAN2: IF2 Command Request	0001h
CAN2IF2DA1	EE4Eh	CAN2: IF2 Data A 1	0000h
CAN2IF2DA2	EE50h	CAN2: IF2 Data A 2	0000h
CAN2IF2DB1	EE52h	CAN2: IF2 Data B 1	0000h
CAN2IF2DB2	EE54h	CAN2: IF2 Data B 2	0000h
CAN2IF2M1	EE44h	CAN2: IF2 Mask 1	FFFFh
CAN2IF2M2	EE46h	CAN2: IF2 Mask 2	FFFFh
CAN2IF2MC	EE4Ch	CAN2: IF2 Message Control	0000h
CAN2IP1	EEA0h	CAN2: Interrupt Pending 1	0000h
CAN2IP2	EEA2h	CAN2: Interrupt Pending 2	0000h
CAN2IR	EE08h	CAN2: Interrupt Register	0000h
CAN2MV1	EEB0h	CAN2: Message Valid 1	0000h
CAN2MV2	EEB2h	CAN2: Message Valid 2	0000h
CAN2ND1	EE90h	CAN2: New Data 1	0000h
CAN2ND2	EE92h	CAN2: New Data 2	0000h
CAN2SR	EE02h	CAN2: Status Register	0000h
CAN2TR	EE0Ah	CAN2: Test Register	00x0h
CAN2TR1	EE80h	CAN2: Transmission Request 1	0000h
CAN2TR2	EE82h	CAN2: Transmission Request 2	0000h
I2CCCR1	EA06h	I2C Clock Control Register 1	0000h
I2CCCR2	EA0Eh	I2C Clock Control Register 2	0000h
I2CCR	EA00h	I2C Control Register	0000h
I2CDR	EA0Ch	I2C Data Register	0000h

Table 72. X-Registers ordered by name (continued)

Name	Physical address	Description	Reset value
I2COAR1	EA08h	I2C Own Address Register 1	0000h
I2COAR2	EA0Ah	I2C Own Address Register 1	0000h
I2CSR1	EA02h	I2C Status Register 1	0000h
I2CSR2	EA04h	I2C Status Register 2	0000h
RTCAH	ED14h	RTC Alarm Register High Byte	xxxxh
RTCAL	ED12h	RTC Alarm Register Low Byte	xxxxh
RTCCON	ED00H	RTC Control Register	0x00h [1]
RTCDH	ED0Ch	RTC Divider Counter High Byte	xxxxh
RTCDL	ED0Ah	RTC Divider Counter Low Byte	xxxxh
RTCH	ED10h	RTC Programmable Counter High Byte	xxxxh
RTCL	ED0Eh	RTC Programmable Counter Low Byte	xxxxh
RTCPH	ED08h	RTC Prescaler Register High Byte	xxxxh
RTCPL	ED06h	RTC Prescaler Register Low Byte	xxxxh
XCLKOUTDIV	EB02h	CLKOUT Divider Control Register	- - 00h
XEMU0	EB76h	XBUS Emulation Register 0 (write only)	xxxxh
XEMU1	EB78h	XBUS Emulation Register 1 (write only)	xxxxh
XEMU2	EB7Ah	XBUS Emulation Register 2 (write only)	xxxxh
XEMU3	EB7Ch	XBUS Emulation Register 3 (write only)	xxxxh
XIR0CLR	EB14h	X-Interrupt 0 Clear Register (write only)	0000h
XIR0SEL	EB10h	X-Interrupt 0 Selection Register	0000h
XIR0SET	EB12h	X-Interrupt 0 Set Register (write only)	0000h
XIR1CLR	EB24h	X-Interrupt 1 Clear Register (write only)	0000h
XIR1SEL	EB20h	X-Interrupt 1 Selection Register	0000h
XIR1SET	EB22h	X-Interrupt 1 Set Register (write only)	0000h
XIR2CLR	EB34h	X-Interrupt 2 Clear Register (write only)	0000h
XIR2SEL	EB30h	X-Interrupt 2 Selection Register	0000h
XIR2SET	EB32h	X-Interrupt 2 Set Register (write only)	0000h
XIR3CLR	EB44h	X-Interrupt 3 Clear Selection Register (write only)	0000h
XIR3SEL	EB40h	X-Interrupt 3 Selection Register	0000h
XIR3SET	EB42h	X-Interrupt 3 Set Selection Register (write only)	0000h
XMISC	EB46h	XBUS Miscellaneous Features Register	0000h
XP1DIDIS	EB36h	PORT1 Digital Disable Register	0000h
XPEREMU	EB7Eh	XPERCON copy for Emulation (write only)	xxxxh

Table 72. X-Registers ordered by name (continued)

Name	Physical address	Description	Reset value
XPICON	EB26h	Extended Port Input Threshold Control Register	- - 00h
XPOLAR	EC04h	XPWM Module Channel Polarity Register	0000h
XPP0	EC20h	XPWM Module Period Register 0	0000h
XPP1	EC22h	XPWM Module Period Register 1	0000h
XPP2	EC24h	XPWM Module Period Register 2	0000h
XPP3	EC26h	XPWM Module Period Register 3	0000h
XPT0	EC10h	XPWM Module Up/Down Counter 0	0000h
XPT1	EC12h	XPWM Module Up/Down Counter 1	0000h
XPT2	EC14h	XPWM Module Up/Down Counter 2	0000h
XPT3	EC16h	XPWM Module Up/Down Counter 3	0000h
XPW0	EC30h	XPWM Module Pulse Width Register 0	0000h
XPW1	EC32h	XPWM Module Pulse Width Register 1	0000h
XPW2	EC34h	XPWM Module Pulse Width Register 2	0000h
XPW3	EC36h	XPWM Module Pulse Width Register 3	0000h
XPWMCON0	EC00h	XPWM Module Control Register 0	0000h
XPWMCON0CLR	EC08h	XPWM Module Clear Control Reg. 0 (write only)	0000h
XPWMCON0SET	EC06h	XPWM Module Set Control Register 0 (write only)	0000h
XPWMCON1	EC02h	XPWM Module Control Register 1	0000h
XPWMCON1CLR	EC0Ch	XPWM Module Clear Control Reg. 0 (write only)	0000h
XPWMCON1SET	EC0Ah	XPWM Module Set Control Register 0 (write only)	0000h
XPWMPORT	EC80h	XPWM Module Port Control Register	0000h
XS1BG	E906h	XASC Baud Rate Generator Reload Register	0000h
XS1CON	E900h	XASC Control Register	0000h
XS1CONCLR	E904h	XASC Clear Control Register (write only)	0000h
XS1CONSET	E902h	XASC Set Control Register (write only)	0000h
XS1PORT	E980h	XASC Port Control Register	0000h
XS1RBUF	E90Ah	XASC Receive Buffer Register	- xxxh
XS1TBUF	E908h	XASC Transmit Buffer Register	0000h
XSSCBR	E80Ah	XSSC Baud Rate Register	0000h
XSSCCON	E800h	XSSC Control Register	0000h

Table 72. X-Registers ordered by name (continued)

Name	Physical address	Description	Reset value
XSSCCONCLR	E804h	XSSC Clear Control Register (write only)	0000h
XSSCCONSET	E802h	XSSC Set Control Register (write only)	0000h
XSSCPORT	E880h	XSSC Port Control Register	0000h
XSSCRB	E808h	XSSC Receive Buffer	xxxxh
XSSCTB	E806h	XSSC Transmit Buffer	0000h

Note: 1. The RTCCON reset value is: 0000 000x 0000 0000b.

26.6 X-Registers ordered by address

The following table lists all X-Bus registers which are implemented in the ST10F276 ordered by their physical address. The Flash control registers are listed in a separate section, in spite of they are also physically mapped on X-Bus memory space. Note that all X-Registers are not bit-addressable.

Table 73. X-Registers ordered by address

Name	Physical address	Description	Reset value
XSSCCON	E800h	XSSC Control Register	0000h
XSSCCONSET	E802h	XSSC Set Control Register (write only)	0000h
XSSCCONCLR	E804h	XSSC Clear Control Register (write only)	0000h
XSSCTB	E806h	XSSC Transmit Buffer	0000h
XSSCRB	E808h	XSSC Receive Buffer	xxxxh
XSSCBR	E80Ah	XSSC Baud Rate Register	0000h
XSSCPORT	E880h	XSSC Port Control Register	0000h
XS1CON	E900h	XASC Control Register	0000h
XS1CONSET	E902h	XASC Set Control Register (write only)	0000h
XS1CONCLR	E904h	XASC Clear Control Register (write only)	0000h
XS1BG	E906h	XASC Baud Rate Generator Reload Register	0000h
XS1TBUF	E908h	XASC Transmit Buffer Register	0000h
XS1RBUF	E90Ah	XASC Receive Buffer Register	- xxxh
XS1PORT	E980h	XASC Port Control Register	0000h
I2CCR	EA00h	I2C Control Register	0000h
I2CSR1	EA02h	I2C Status Register 1	0000h
I2CSR2	EA04h	I2C Status Register 2	0000h
I2CCCR1	EA06h	I2C Clock Control Register 1	0000h

Table 73. X-Registers ordered by address (continued)

Name	Physical address	Description	Reset value
I2COAR1	EA08h	I2C Own Address Register 1	0000h
I2COAR2	EA0Ah	I2C Own Address Register 1	0000h
I2CDR	EA0Ch	I2C Data Register	0000h
I2CCCR2	EA0Eh	I2C Clock Control Register 2	0000h
XCLKOUTDIV	EB02h	CLKOUT Divider Control Register	- - 00h
XIR0SEL	EB10h	X-Interrupt 0 Selection Register	0000h
XIR0SET	EB12h	X-Interrupt 0 Set Register (write only)	0000h
XIR0CLR	EB14h	X-Interrupt 0 Clear Register (write only)	0000h
XIR1SEL	EB20h	X-Interrupt 1 Selection Register	0000h
XIR1SET	EB22h	X-Interrupt 1 Set Register (write only)	0000h
XIR1CLR	EB24h	X-Interrupt 1 Clear Register (write only)	0000h
XPICON	EB26h	Extended Port Input Threshold Control Register	- - 00h
XIR2SEL	EB30h	X-Interrupt 2 Selection Register	0000h
XIR2SET	EB32h	X-Interrupt 2 Set Register (write only)	0000h
XIR2CLR	EB34h	X-Interrupt 2 Clear Register (write only)	0000h
XP1DIDIS	EB36h	PORT1 Digital Disable Register	0000h
XIR3SEL	EB40h	X-Interrupt 3 Selection Register	0000h
XIR3SET	EB42h	X-Interrupt 3 Set Selection Register (write only)	0000h
XIR3CLR	EB44h	X-Interrupt 3 Clear Selection Register (write only)	0000h
XMISC	EB46h	XBUS Miscellaneous Features Register	0000h
XEMU0	EB76h	XBUS Emulation Register 0 (write only)	xxxxh
XEMU1	EB78h	XBUS Emulation Register 1 (write only)	xxxxh
XEMU2	EB7Ah	XBUS Emulation Register 2 (write only)	xxxxh
XEMU3	EB7Ch	XBUS Emulation Register 3 (write only)	xxxxh
XPEREMU	EB7Eh	XPERCON copy for Emulation (write only)	xxxxh
XPWMCON0	EC00h	XPWM Module Control Register 0	0000h
XPWMCON1	EC02h	XPWM Module Control Register 1	0000h
XPOLAR	EC04h	XPWM Module Channel Polarity Register	0000h
XPWMCON0SET	EC06h	XPWM Module Set Control Register 0 (write only)	0000h
XPWMCON0CLR	EC08h	XPWM Module Clear Control Reg. 0 (write only)	0000h

Table 73. X-Registers ordered by address (continued)

Name	Physical address	Description	Reset value
XPWMCON1SET	EC0Ah	XPWM Module Set Control Register 0 (write only)	0000h
XPWMCON1CLR	EC0Ch	XPWM Module Clear Control Reg. 0 (write only)	0000h
XPT0	EC10h	XPWM Module Up/Down Counter 0	0000h
XPT1	EC12h	XPWM Module Up/Down Counter 1	0000h
XPT2	EC14h	XPWM Module Up/Down Counter 2	0000h
XPT3	EC16h	XPWM Module Up/Down Counter 3	0000h
XPP0	EC20h	XPWM Module Period Register 0	0000h
XPP1	EC22h	XPWM Module Period Register 1	0000h
XPP2	EC24h	XPWM Module Period Register 2	0000h
XPP3	EC26h	XPWM Module Period Register 3	0000h
XPW0	EC30h	XPWM Module Pulse Width Register 0	0000h
XPW1	EC32h	XPWM Module Pulse Width Register 1	0000h
XPW2	EC34h	XPWM Module Pulse Width Register 2	0000h
XPW3	EC36h	XPWM Module Pulse Width Register 3	0000h
XPWMPORT	EC80h	XPWM Module Port Control Register	0000h
RTCCON	ED00H	RTC Control Register	0x00h
RTCPL	ED06h	RTC Prescaler Register Low Byte	xxxxh
RTCPH	ED08h	RTC Prescaler Register High Byte	xxxxh
RTCDL	ED0Ah	RTC Divider Counter Low Byte	xxxxh
RTCDH	ED0Ch	RTC Divider Counter High Byte	xxxxh
RTCL	ED0Eh	RTC Programmable Counter Low Byte	xxxxh
RTCH	ED10h	RTC Programmable Counter High Byte	xxxxh
RTCAL	ED12h	RTC Alarm Register Low Byte	xxxxh
RTCAH	ED14h	RTC Alarm Register High Byte	xxxxh
CAN2CR	EE00h	CAN2: CAN Control Register	0001h
CAN2SR	EE02h	CAN2: Status Register	0000h
CAN2EC	EE04h	CAN2: Error Counter	0000h
CAN2BTR	EE06h	CAN2: Bit Timing Register	2301h
CAN2IR	EE08h	CAN2: Interrupt Register	0000h
CAN2TR	EE0Ah	CAN2: Test Register	00x0h
CAN2BRPER	EE0Ch	CAN2: BRP Extension Register	0000h
CAN2IF1CR	EE10h	CAN2: IF1 Command Request	0001h
CAN2IF1CM	EE12h	CAN2: IF1 Command Mask	0000h

Table 73. X-Registers ordered by address (continued)

Name	Physical address	Description	Reset value
CAN2IF1M1	EE14h	CAN2: IF1 Mask 1	FFFFh
CAN2IF1M2	EE16h	CAN2: IF1 Mask 2	FFFFh
CAN2IF1A1	EE18h	CAN2: IF1 Arbitration 1	0000h
CAN2IF1A2	EE1Ah	CAN2: IF1 Arbitration 2	0000h
CAN2IF1MC	EE1Ch	CAN2: IF1 Message Control	0000h
CAN2IF1DA1	EE1Eh	CAN2: IF1 Data A 1	0000h
CAN2IF1DA2	EE20h	CAN2: IF1 Data A 2	0000h
CAN2IF1DB1	EE22h	CAN2: IF1 Data B 1	0000h
CAN2IF1DB2	EE24h	CAN2: IF1 Data B 2	0000h
CAN2IF2CR	EE40h	CAN2: IF2 Command Request	0001h
CAN2IF2CM	EE42h	CAN2: IF2 Command Mask	0000h
CAN2IF2M1	EE44h	CAN2: IF2 Mask 1	FFFFh
CAN2IF2M2	EE46h	CAN2: IF2 Mask 2	FFFFh
CAN2IF2A1	EE48h	CAN2: IF2 Arbitration 1	0000h
CAN2IF2A2	EE4Ah	CAN2: IF2 Arbitration 2	0000h
CAN2IF2MC	EE4Ch	CAN2: IF2 Message Control	0000h
CAN2IF2DA1	EE4Eh	CAN2: IF2 Data A 1	0000h
CAN2IF2DA2	EE50h	CAN2: IF2 Data A 2	0000h
CAN2IF2DB1	EE52h	CAN2: IF2 Data B 1	0000h
CAN2IF2DB2	EE54h	CAN2: IF2 Data B 2	0000h
CAN2TR1	EE80h	CAN2: Transmission Request 1	0000h
CAN2TR2	EE82h	CAN2: Transmission Request 2	0000h
CAN2ND1	EE90h	CAN2: New Data 1	0000h
CAN2ND2	EE92h	CAN2: New Data 2	0000h
CAN2IP1	EEA0h	CAN2: Interrupt Pending 1	0000h
CAN2IP2	EEA2h	CAN2: Interrupt Pending 2	0000h
CAN2MV1	EEB0h	CAN2: Message Valid 1	0000h
CAN2MV2	EEB2h	CAN2: Message Valid 2	0000h
CAN1CR	EF00h	CAN1: CAN Control Register	0001h
CAN1SR	EF02h	CAN1: Status Register	0000h
CAN1EC	EF04h	CAN1: Error Counter	0000h
CAN1BTR	EF06h	CAN1: Bit Timing Register	2301h
CAN1IR	EF08h	CAN1: Interrupt Register	0000h
CAN1TR	EF0Ah	CAN1: Test Register	00x0h

Table 73. X-Registers ordered by address (continued)

Name	Physical address	Description	Reset value
CAN1BRPER	EF0Ch	CAN1: BRP Extension Register	0000h
CAN1IF1CR	EF10h	CAN1: IF1 Command Request	0001h
CAN1IF1CM	EF12h	CAN1: IF1 Command Mask	0000h
CAN1IF1M1	EF14h	CAN1: IF1 Mask 1	FFFFh
CAN1IF1M2	EF16h	CAN1: IF1 Mask 2	FFFFh
CAN1IF1A1	EF18h	CAN1: IF1 Arbitration 1	0000h
CAN1IF1A2	EF1Ah	CAN1: IF1 Arbitration 2	0000h
CAN1IF1MC	EF1Ch	CAN1: IF1 Message Control	0000h
CAN1IF1DA1	EF1Eh	CAN1: IF1 Data A 1	0000h
CAN1IF1DA2	EF20h	CAN1: IF1 Data A 2	0000h
CAN1IF1DB1	EF22h	CAN1: IF1 Data B 1	0000h
CAN1IF1DB2	EF24h	CAN1: IF1 Data B 2	0000h
CAN1IF2CR	EF40h	CAN1: IF2 Command Request	0001h
CAN1IF2CM	EF42h	CAN1: IF2 Command Mask	0000h
CAN1IF2M1	EF44h	CAN1: IF2 Mask 1	FFFFh
CAN1IF2M2	EF46h	CAN1: IF2 Mask 2	FFFFh
CAN1IF2A1	EF48h	CAN1: IF2 Arbitration 1	0000h
CAN1IF2A2	EF4Ah	CAN1: IF2 Arbitration 2	0000h
CAN1IF2MC	EF4Ch	CAN1: IF2 Message Control	0000h
CAN1IF2DA1	EF4Eh	CAN1: IF2 Data A 1	0000h
CAN1IF2DA2	EF50h	CAN1: IF2 Data A 2	0000h
CAN1IF2DB1	EF52h	CAN1: IF2 Data B 1	0000h
CAN1IF2DB2	EF54h	CAN1: IF2 Data B 2	0000h
CAN1TR1	EF80h	CAN1: Transmission Request 1	0000h
CAN1TR2	EF82h	CAN1: Transmission Request 2	0000h
CAN1ND1	EF90h	CAN1: New Data 1	0000h
CAN1ND2	EF92h	CAN1: New Data 2	0000h
CAN1IP1	EFA0h	CAN1: Interrupt Pending 1	0000h
CAN1IP2	EFA2h	CAN1: Interrupt Pending 2	0000h
CAN1MV1	EFB0h	CAN1: Message Valid 1	0000h
CAN1MV2	EFB2h	CAN1: Message Valid 2	0000h

26.7 Flash registers ordered by name

The following table lists all Flash Control Registers which are implemented in the ST10F276 ordered by their name. Note that as they are physically mapped on the X-Bus, these registers are not bit-addressable.

Table 74. Flash registers ordered by name

Name	Physical address	Description	Reset value
FARH	0x000E 0012	Flash Address Register High	0000h
FARL	0x000E 0010	Flash Address Register Low	0000h
FCR0H	0x000E 0002	Flash Control Register 0 - High	0000h
FCR0L	0x000E 0000	Flash Control Register 0 - Low	0000h
FCR1H	0x000E 0006	Flash Control Register 1 - High	0000h
FCR1L	0x000E 0004	Flash Control Register 1 - Low	0000h
FDR0H	0x000E 000A	Flash Data Register 0 - High	FFFFh
FDR0L	0x000E 0008	Flash Data Register 0 - Low	FFFFh
FDR1H	0x000E 000E	Flash Data Register 1 - High	FFFFh
FDR1L	0x000E 000C	Flash Data Register 1 - Low	FFFFh
FER	0x000E 0014	Flash Error Register	0000h
FNVAPR0	0x000E DFB8	Flash Non Volatile Access Protection Reg. 0	ACFFh
FNVAPR1H	0x000E DFBE	Flash Non Volatile Access Protection Reg. 1 - High	FFFFh
FNVAPR1L	0x000E DFBC	Flash Non Volatile Access Protection Reg. 1 - Low	FFFFh
FNVWPIRH	0x000E DFB6	Flash Non Volatile Protection I Reg. High	FFFFh
FNVWPIRL	0x000E DFB4	Flash Non Volatile Protection I Reg. Low	FFFFh
FNVWPXRH	0x000E DFB2	Flash Non Volatile Protection X Reg. High	FFFFh
FNVWPXRL	0x000E DFB0	Flash Non Volatile Protection X Reg. Low	FFFFh
XFICR	0x000E E000	XFlash Interface Control Register	000Fh

26.8 Flash registers ordered by address

The following table lists all Flash Control Registers which are implemented in the ST10F276 ordered by their physical address. Note that as they are physically mapped on the X-Bus, these registers are not bit-addressable.

Table 75. Flash registers ordered by address

Name	Physical address	Description	Reset value
FCR0L	0x000E 0000	Flash Control Register 0 - Low	0000h
FCR0H	0x000E 0002	Flash Control Register 0 - High	0000h
FCR1L	0x000E 0004	Flash Control Register 1 - Low	0000h

Table 75. Flash registers ordered by address (continued)

Name	Physical address	Description	Reset value
FCR1H	0x000E 0006	Flash Control Register 1 - High	0000h
FDR0L	0x000E 0008	Flash Data Register 0 - Low	FFFFh
FDR0H	0x000E 000A	Flash Data Register 0 - High	FFFFh
FDR1L	0x000E 000C	Flash Data Register 1 - Low	FFFFh
FDR1H	0x000E 000E	Flash Data Register 1 - High	FFFFh
FARL	0x000E 0010	Flash Address Register Low	0000h
FARH	0x000E 0012	Flash Address Register High	0000h
FER	0x000E 0014	Flash Error Register	0000h
FNVWPXRL	0x000E DFB0	Flash Non Volatile Protection X Reg. Low	FFFFh
FNVWPXRH	0x000E DFB2	Flash Non Volatile Protection X Reg. High	FFFFh
FNVWPIRL	0x000E DFB4	Flash Non Volatile Protection I Reg. Low	FFFFh
FNVWPIRH	0x000E DFB6	Flash Non Volatile Protection I Reg. High	FFFFh
FNVAPR0	0x000E DFB8	Flash Non Volatile Access Protection Reg. 0	ACFFh
FNVAPR1L	0x000E DFBC	Flash Non Volatile Access Protection Reg. 1 - Low	FFFFh
FNVAPR1H	0x000E DFBE	Flash Non Volatile Access Protection Reg. 1 - High	FFFFh
XFICR	0x000E E000	XFlash Interface Control Register	000Fh

26.9 Special notes

PEC pointer registers

The source and destination pointers for the peripheral event controller are mapped to a special area within the IRAM. 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 *Section 5: Interrupt and trap functions on page 96*.

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).

```
MOVR5, DP3;GPR access via SFR area
EXTR#1
MOVR5, 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.

26.10 Identification registers

The ST10F276 has four Identification registers, mapped in ESFR space. These registers contain:

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

IDMANUF (F07Eh / 3Fh) ESFR Reset Value: 0403h

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

Bit	Function
MANUF	**Manufacturer Identifier** 020h: STMicroelectronics manufacturer (JTAG worldwide normalization).

IDCHIP (F07Ch / 3Eh) ESFR Reset Value: 114xh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCONF		IDCHIP										REVID			
R		R										R			

Bit	Function
PCONF	**Peripheral Configuration** ‘00’: (E) Enhanced (ST10F276x) ‘01’: (B) Basic ‘10’: (D) Dedicated ‘11’: *reserved*
IDCHIP	**Device Identifier** 114h: ST10F276 Identifier (276 in decimal format).
REVID	**Device Revision Identifier** Xh: According to revision number (1: for Axx steps, 2: for Bxx, 3: for Cxx and so on)

IDMEM (F07Ah / 3Dh) ESFR Reset Value:30D0h

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

Bit	Function
MEMSIZE	**Internal Memory Size** Internal Memory size is 4 x (MEMSIZE) (in Kbyte) 0D0h for ST10F276 (832 Kbytes)
MEMTYP	**Internal Memory Type** '0h': ROM-Less '1h': (M) ROM memory '2h': (S) Standard Flash memory '3h': (H) High Performance Flash memory (ST10F276x) '4h...Fh': *reserved*

IDPROG (F078h / 3Ch) ESFR Reset Value: 0040h

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

Bit	Function
PROGVDD	**Programming V_{DD} Voltage** V_{DD} voltage when programming EPROM or Flash devices is calculated using the following formula: V_{DD} = 20 x [PROGVDD] / 256 (volts) - 40h for ST10F276 (5V).
PROGVPP	**Programming V_{PP} Voltage** (no need of external V_{PP}) - 00h

Note: 1. All identification words are read only registers.

The values written inside different Identification Register bits are valid only after the Flash initialization phase is completed. When code execution is started from internal memory (pin $\overline{EA}$ held high during reset), the Flash has certainly completed its initialization, so the bits of Identification Registers are immediately ready to be read out. On the contrary, when code execution is started from external memory (pin $\overline{EA}$ held low during reset), the Flash initialization is not yet completed, so the bits of Identification Registers are not ready. The user can poll bits 15 and 14 of IDMEM register: when both bits are read low, the Flash initialization is complete, so all Identification Register bits are correct.

Before Flash initialization completion, the default setting of the different Identification Registers are the following:

IDMANUF 0403h

IDCHIP 114xh (with x representing the silicon revision number)

IDMEM F0D0h

IDPROG 0040h

27 System programming

Constructs for modularity, loops, and context switching have been built into the ST10F276 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 ST10F276.

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 ST10F276:

Substituted instruction	ST10F276 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 ST10F276 provides a unified memory architecture and its on-chip hardware automatically detects accesses to IRAM, GPRs, and SFRs.

Multiplication and division

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

The MDRIU flag (Multiply or Divide Register In Use) in register MDC is set whenever either half of this register is written to or when a multiply/divide instruction is started. It is cleared whenever the MDL register is read.

Because an interrupt can be acknowledged before the contents of register MD are saved, this flag is required to alert interrupt routines, which require the use of the multiply/divide hardware, so they can preserve register MD.

This register, however, only needs to be saved when an interrupt routine requires use of the MD register and a previous task has not saved the current result. This flag is easily tested by the Jump-on bit instructions.

Multiplication or division is simply performed by specifying the correct (signed or unsigned) version of the multiply or divide instruction. The result is then stored in register MD.

The overflow flag (V) is set if the result from a multiply or divide instruction is greater than 16-bit. This flag can be used to determine whether both word halves must be transferred from register MD.

The high portion of register MD (MDH) must be moved into the register file or memory first, in order to ensure that the MDRIU flag reflects the correct state.

The following instruction sequence performs an unsigned 16 by 16-bit multiplication:

```
            ...
SAVE:       JNB     MDRIU, START      ;Test if MD was in use.
            SCXT    MDC, #0010H       ;Save and clear control register, leaving MDRIU set
                                      ;(only req for interrupted multiply/divide instructions)
            BSET    SAVED             ;Indicate the save operation
            PUSH    MDH               ;Save previous MD contents...
            PUSH    MDL               ;...on system stack
START:      MULU    R1, R2            ;Multiply 16*16 unsigned, Sets MDRIU
            JMPR    cc_NV, COPYL      ;Test for only 16 Bit result
            MOV     R3, MDH           ;Move high portion of MD
COPYL:      MOV     R4, MDL           ;Move low portion of MD, Clears MDRIU
RESTORE:    JNB     SAVED, DONE       ;Test if MD registers were saved
            POP     MDL               ;Restore registers
            POP     MDH
            POP     MDC
            BCLR    SAVED             ;Multiplication is completed, program continues
DONE:       ...
```

The above save sequence and the restore sequence after COPYL are only required if the current routine could have interrupted a previous routine which contained a MUL or DIV instruction. Register MDC is also saved because it is possible that a previous routine's Multiply or Divide instruction was interrupted while in progress. In this case the information about how to restart the instruction is contained in this register. Register MDC must be cleared to be correctly initialized for a subsequent multiplication or division. The old MDC contents must be popped from the stack before the RETI instruction is executed.

For a division the user must first move the dividend into the MD register. If a 16 by 16-bit division is specified, only the low portion of register MD must be loaded.

The result is also stored into register MD. The low portion (MDL) contains the integer result of the division, while the high portion (MDH) contains the remainder.

The following instruction sequence performs a 32 by 16-bit division:

```
            MOV     MDH, R1           ;Move dividend to MD register. Sets MDRIU
            MOV     MDL, R2           ;Move low portion to MD
            DIV     R3                ;Divide 32/16 signed, R3 holds the divisor
            JMPR    cc_V, ERROR       ;Test for divide overflow
            MOV     R3, MDH           ;Move remainder to R3
            MOV     R4, MDL           ;Move integer result to R4. Clears MDRIU
```

Whenever a multiply or divide instruction is interrupted while in progress, the address of the interrupted instruction is pushed onto the stack and the MULIP flag in the PSW of the interrupting routine is set. When the interrupt routine is exited with the RETI instruction, this bit is implicitly tested before the old PSW is popped from the stack. If MULIP = '1' the multiply/divide instruction is re-read from the location popped from the stack (return address) and will be completed after the RETI instruction has been executed.

*Note: The MULIP flag is part of the **context of the interrupted task**. When the interrupting routine does not return to the interrupted task (for example when a scheduler switches to another task) the MULIP flag must be set or cleared according to the context of the task that is switched to.*

BCD calculations

No direct support for BCD calculations is provided in the ST10F276. 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.

27.1 Stack operations

The ST10F276 supports two types of stacks. The system stack is used implicitly by the controller and is located in the IRAM. 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 IRAM 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 IRAM via hardware. This virtual stack area covers all possible locations that SP can point to, from 00’F000h through 00’FFFEh. STKOV and STKUN accept the same 4 Kbyte address range.

The size of the physical stack area within the IRAM that effectively is used for standard stack operations is defined via bit-field STKSZ in register SYSCON (see below).

Table 76. Stack size selection

(STKSZ)	Stack size (words)	IRAM addresses (words) of physical stack	Significant bit of stack pointer SP
0 0 0 b	256	00’FBFEh...00’FA00h (Default after Reset)	SP.8...SP.0
0 0 1 b	128	00’FBFEh...00’FB00h	SP.7...SP.0
0 1 0 b	64	00’FBFEh...00’FB80h	SP.6...SP.0
0 1 1 b	32	00’FBFEh...00’FBC0h	SP.5...SP.0
1 0 0 b	512	00’FBFEh...00’F800h (not for 1 Kbyte IRAM)	SP.9...SP.0

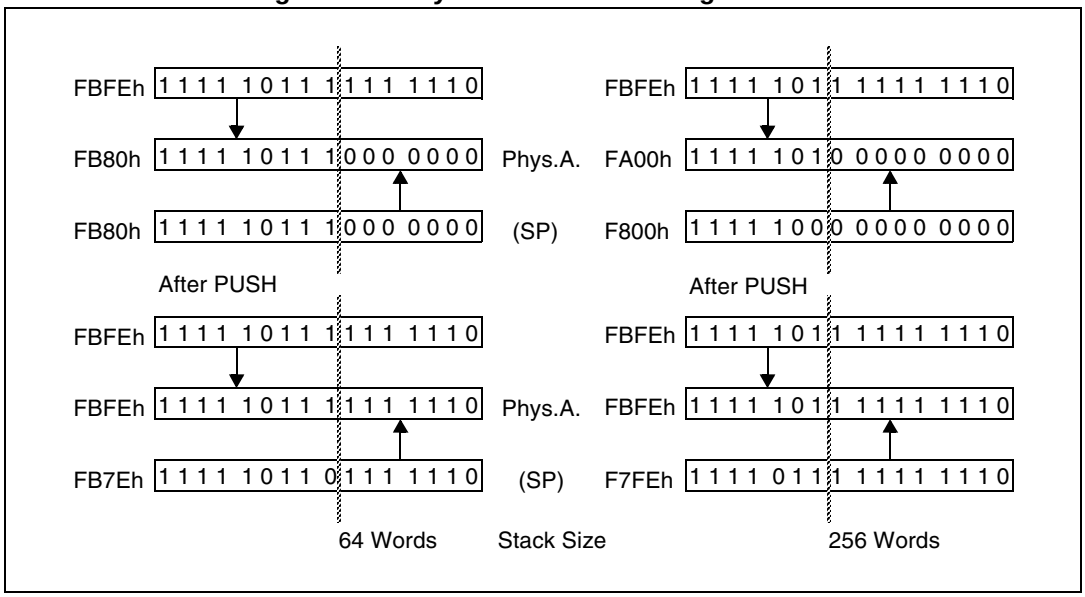
Table 76. Stack size selection (continued)

(STKSZ)	Stack size (words)	IRAM addresses (words) of physical stack	Significant bit of stack pointer SP
1 0 1 b	---	Reserved. Do not use this combination.	---
1 1 0 b	---	Reserved. Do not use this combination.	---
1 1 1 b	1024	00'FDFEh...00'F600h (Note: No circular stack)	SP.11...SP.0

The virtual stack addresses are transformed to physical stack addresses by concatenating the significant bit of the stack pointer register SP (see table) with the complementary most significant bit of the upper limit of the physical stack area (00'FBFEh). This transformation is done via hardware (see *Figure 214*).

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 214. Physical stack address generation



The following example demonstrates the circular stack mechanism which is also an effect of this virtual stack mapping: First, register R1 is pushed onto the lowest physical stack location according to the selected maximum stack size. With the following instruction, register R2 will be pushed onto the highest physical stack location although the SP is decremented by 2 as for the previous push operation.

```
MOV SP, #0F802h ; Set SP before last entry of physical stack of 256 Words
...             ; (SP) = F802h: Physical stack address = FA02h
PUSH R1         ; (SP) = F800h: Physical stack address = FA00h
PUSH R2         ; (SP) = F7FEh: Physical stack address = FBFEh
```

The effect of the address transformation is that the physical stack addresses wrap around from the end of the defined area to its beginning. When flushing and filling the internal stack,

this circular stack mechanism only requires to move that portion of stack data which is really to be re-used (the upper part of the defined stack area) instead of the whole stack area. Stack data that remain in the lower part of the internal stack need not be moved by the distance of the space being flushed or filled, as the stack pointer automatically wraps around to the beginning of the freed part of the stack area.

Note: This circular stack technique is applicable for stack sizes of 32 to 512 words (STKSZ = '000b' to '100b'), it does not work with option STKSZ = '111b', which uses the complete IRAM 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 IRAM (bit-field STKSZ in register SYSCON).
- Define two pointers, which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.
- Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six words (for the reserved space to store two interrupt entries).

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

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

Linear stack

The ST10F276 also offers a linear stack option (STKSZ = '111b'), where the system stack may use the complete IRAM 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 IRAM and therefore may only use the address range 00'F600h to 00'FDFEh. It is the user's responsibility to restrict the system stack to the IRAM 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.

27.2 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.

27.3 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.

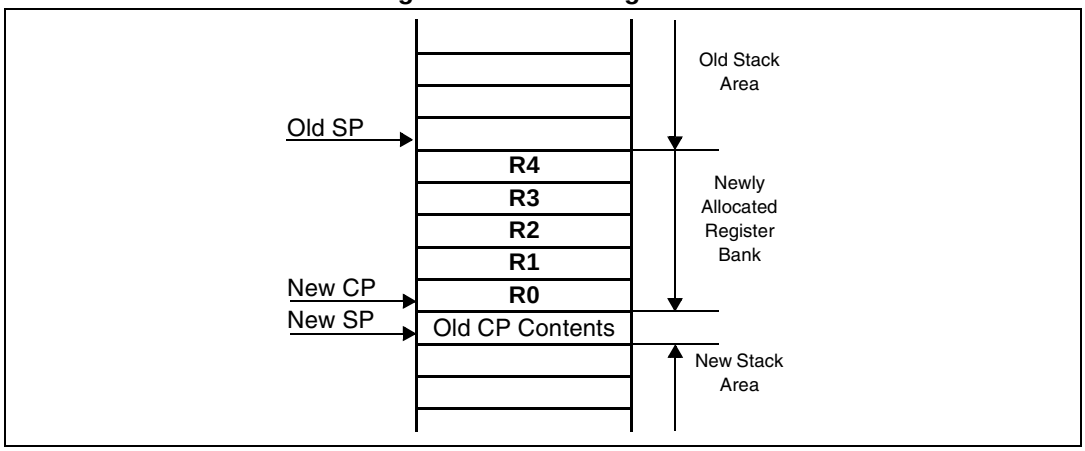
Saving and restoring of registers: To provide local registers, the contents of the registers which are required for use by the subroutine can be pushed onto the stack and the previous values be popped before returning to the calling routine. This is the most common technique used today and it does provide 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 215. Local registers



Software to provide the local register bank for the example in *Figure 215* 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
```

27.4 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.

Two examples below illustrate searching of ordered and non-ordered tables, respectively:

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

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

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

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

27.5 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 ST10F276 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.

27.6 Floating point support

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

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

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

27.7 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.

27.8 Inseparable instruction sequences

The instructions of the ST10F276 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** ($\overline{\text{NMI}}$, stack overflow/underflow) are disabled. A **Class B Trap** (such as illegal opcode or illegal bus access), 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
```

27.9 Overriding the DPP addressing mechanism

The standard mechanism to access data locations uses one of the four data page pointers (DPPx), which selects a 16 Kbytes data page, and a 14-bit offset within this data page. The four DPPs allow immediate access to up to 64 Kbytes of data. In applications with big data arrays, especially in HLL applications using large memory models, this may require frequent reloading of the DPPs, even for single accesses.

The EXTP (extend page) instruction allows switching to an arbitrary data page for 1...4 instructions without having to change the current DPPs.

```
EXAMPLE:  EXTP     R15, #1     ; The override page number is stored in R15
          MOV      R0, [R14]   ; The (14 Bit) page offset is stored in R14
          MOV      R1, [R13]   ; This instruction uses the standard DPP
                               scheme!
```

The EXTS (extend segment) instruction allows switching to a 64 Kbyte segment oriented data access scheme for 1...4 instructions without having to change the current DPPs. In this case all 16 bits of the operand address are used as segment offset, with the segment taken from the EXTS instruction. This greatly simplifies address calculation with continuous data like huge arrays in “C”.

```
EXAMPLE:  EXTS  #15, #1   ; The override seg. is #15 (0F`0000h...0F`FFFFh)
          MOV   R0, [R14] ; The (16 Bit) segment offset is stored in R14
          MOV   R1, [R13] ; This instruction uses the standard DPP scheme!
```

Note: Instructions EXTP and EXTS inhibit interrupts the same way as ATOMIC.

Short addressing in the extended SFR (ESFR) space

The short addressing modes of the ST10F276 (REG or bitOFF) implicitly access the SFR space. The additional ESFR space would have to be accessed via long addressing modes (MEM or [Rw]).

The EXTR (extend register) instruction redirects accesses in short addressing modes to the ESFR space for 1...4 instructions, so the additional registers can be accessed this way, too.

The EXTPR and EXTSR instructions combine the DPP override mechanism with the redirection to the ESFR space using a single instruction.

Note: Instructions EXTR, EXTPR and EXTSR inhibit interrupts the same way as ATOMIC.
The switching to the ESFR area and data page overriding is checked by the development tools or handled automatically.

Nested locked sequences

Each of the described extension instruction and the ATOMIC instruction starts an internal “extension counter” counting the effected instructions. When another extension or ATOMIC instruction is contained in the current locked sequence this counter is restarted with the value of the new instruction. This allows the construction of locked sequences longer than four 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.

27.10 Handling the internal Flash

The ST10F276 provides and controls up 832 Kbytes of internal on-chip Flash memory that may store code as well as data. Access to this internal Flash area is controlled during the reset configuration and via software.

Configuration during reset

The default memory configuration of the ST10F276 Memory is determined by the state of the $\overline{EA}$ pin at reset. This value is stored in the Internal ROM Enable bit (named ROMEN) of the SYSCON register.

When the $\overline{EA}$ pin is high during reset (default value), the internal Flash is globally enabled and the first 32 Kbytes are mapped in segment ‘0’. The first instructions are fetched from the internal Flash from locations 00’0000h.

When the $\overline{EA}$ pin is low during reset (ROMEN = 0), the internal Flash is disabled and external memory is used for startup control.

Mapping the internal Flash area

When internal Flash is disabled on reset the first instructions are fetched from external memory locations 00’0000h. The Flash memory can later be enabled by setting the ROMEN bit of SYSCON to 1. The code performing this setting must not run from a segment of the external memory to be replaced by a segment of the Flash memory, otherwise unexpected behavior may occur.

For example, if external memory code is located in the first 32 Kbytes of segment 0, the first 32 Kbytes of the Flash must then be enabled in segment 1. This is done by setting the ROMS1 bit of SYSCON to 0 before or simultaneously with setting of ROMEN bit. This must be done in the externally supplied program before the execution of the EINIT instruction.

If program execution starts from external memory, but access to the Flash memory mapped in segment 0 is later required, then the code that performs the setting of ROMEN bit must be executed either in the segment 0 but above address 00’8000h, or from the on-chip RAM.

Bit ROMS1 only affects the mapping of the first 32 Kbytes of the IFlash memory. All other parts of the IFlash memory (addresses 01’8000h - 08’FFFFh) remain unaffected.

The SGTDIS Segmentation Disable / Enable must also be set to 0 to allow the use of the full on-chip Flash memory in addition to the external boot memory. The correct procedure on changing the segmentation registers must also be observed to prevent an unwanted trap condition:

- Instructions that configure the internal memory must only be executed from external memory or from the IRAM.
- An Absolute Inter-Segment Jump (JMPS) instruction must be executed after Flash enabling, to the next instruction, even if this next instruction is located in the consecutive address.
- Whenever the internal Memory is disabled, enabled or re-mapped, the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal memory and/or external memory.

When starting from external memory, the interrupt/trap vector table, which uses locations 00’0000h through 00’01FFh, of the external memory and may therefore be modified, so the system software may now change interrupt/trap handlers according to the current condition of the system.

The internal Flash can still be used for fixed software routines such as I/O drivers, math libraries, application specific invariant routines, or tables. This combines the advantage of an integrated non-volatile memory with the advantage of a flexible, adaptable software system.

Enabling and disabling the internal Flash area after reset

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

If the internal Flash only contains the start-up code and/or test software, the system may be booted from internal Flash, which may then be disabled, after the software has switched to executing from external memory, in order to free the address space occupied by the internal Flash area, which is now unnecessary.

27.11 Pits, traps and mines

Although handling the internal 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 ST10F276 and it must be made sure that it never runs out of it.

The following precautions help to take advantage of the methods mentioned above without jeopardizing system security.

Internal Flash access after reset: When the first instructions are to be fetched from internal Flash (EA = '1'), the memory must contain a valid reset vector and valid code at its destination.

Mapping the internal Flash to segment 1: Due to instruction pipelining, any new Flash mapping will at the earliest become valid for the second instruction after the instruction which has changed the Flash mapping. To enable accesses to the Flash after mapping a branch to the newly selected Flash area (JMPS) and reloading of all data page pointers is required.
This also applies to re-mapping the internal Flash to segment 0.

Enabling the internal Flash after reset: When enabling the internal Flash after having booted the system from external memory, note that the ST10F276 will then access the internal Flash using the current segment offset, rather than accessing external memory.

Disabling the internal Flash after reset: When disabling the internal Flash after having booted the system from there, note that the ST10F276 will not access external memory before a jump to segment 0 (in this case) is executed.

General rules

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

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

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

Appendix A Abbreviations

The following abbreviations and acronyms are used in this user manual:

ABM	Alternate Boot Mode
ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Controller
BRG	Baud Rate Generator
CAN	Controller Area Network (License Bosch^{TM})
CAPCOM	CAPture and COMpare unit
CISC	Complex Instruction Set Computing
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
DLC	Data Length Code
EBC	External Bus Controller
ESFR	Extended Special Function Register
FIFO	First In - First Out
Flash	Non-volatile memory that may be electrically erased
FSM	Finite State Machine
GPR	General Purpose Register
GPT	General Purpose Timer unit
HLL	High Level Language
IRAM	On-chip Internal Dual Port RAM
I/O	Input / Output
I^2C	Inter Integrated Circuit
MCU	Micro-Controller Unit
PEC	Peripheral Event Controller
PLA	Programmable Logic Array
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RTC	Real Time Clock
SFR	Special Function Register
SSC	Synchronous Serial Controller
TOS	Top Of Stack
XBUS	Internal representation of the External Bus
XRAM	On-chip extension RAM

Appendix B Document references

1. 16-bit MCU with MAC unit 832 Kbyte Flash memory and 68 Kbyte RAM (ST10F276E, DocID12303)
2. 16-bit MCU with MAC unit, 832 Kbyte Flash memory and 68 Kbyte RAM (ST10F276Z5, DocID12448)

Revision history

Table 77. Document revision history

Date	Revision	Changes
21-Feb-2003	0.1	It is the first official release of the document, just for ST internal review only.
1-Sep-2003	0.2	Many modifications according to internal review procedure.
1-Oct-2003	0.3	Minor modifications according to internal review procedure.
7-Jan-2004	0.4	*Table 58 on page 411* updated with correct registers address of CAN2 module (EExxh instead of EFxxh). *Table 61 on page 429* updated adding BRP+BRPE extedend prescaler option. *System clock tolerance range on page 457*: title and content updated to detail tolerance quations vs. PLL jitter effect. *Calculation of the bit timing parameters on page 461*: examples of bit timing calculation updated with PLL jitter effect. *Section 23.8: Reset application examples on page 492*: examples of bidirectional reset added. Section 25.2.3: removed reference to oscillator frequency range (4 to 8 MHz).

Table 77. Document revision history (continued)

Date	Revision	Changes
19-Mar-2004	0.5	*Figure 4 on page 41* and *Figure 6 on page 45* updated: B0F6 into B0F7. XPER-SHARE Mode limits added in paragraph *Section 2.4.1: XRAM access via external masters on page 50*. Bit XPER-SHARE in SYSCON: description updated. Note added in the paragraph *Section 5.7: X-peripheral interrupt on page 117*. *Figure 74 on page 214* and *Figure 75 on page 216* updated: B0F6 into B0F7. Modified note for ADC minimum conversion time limitation - *Note: on page 382*. EML acronym replaced with extended format: Error Management Logic in *Section 21.7.1: Software initialization on page 418* and *Section : Status Register on page 425*. VREGOFF description updated. Note about status of $\overline{\text{RTSIN}}$ pin at Power-On added - *Note: on page 474*. RP0H description updated (bit CLKCFG: PLL multiplication factors x8 and x10 swapped). Table Header updated replacing LP OSC with 32 kHz - *Table 65 on page 497*. *Section 24.2: Power down mode on page 504*: Voltage Regulator in Power Down description updated. 32 kHz oscillator: removed low-power statement - *Section 24.3: Stand-by mode on page 509*. Low Power oscillator substituted with 32 kHz oscillator - *Section 24.3.2: Exiting stand-by mode on page 511*. PICON and XPICON registers added in the tables in *Section 26: Register set on page 515*. XPWMCON0SET/CLR and XPWMCON1SET/CLR register names corrected at *Section 26: Register set on page 515*. Note about bit addressability of Flash Control Registers updated in *Section 26.7: Flash registers ordered by name on page 541* and *Section 26.8: Flash registers ordered by address on page 541*. *Section 27: System programming on page 545* updated replacing all references to internal ROM with internal Flash, and external ROM with external memory.

Table 77. Document revision history (continued)

Date	Revision	Changes
24-Jan-2005	0.6	Sections 10 to 16 re-ordered as in the original version (bad order in rev. 1.5 only). *Section 7: Dedicated pins on page 179*: added V_{AREF} and V_{AGND} as dedicated pins in the table. *Section 6.10: Port8*: *Figure 53 on page 176* updated - unidirectional arrows on XPWM alternate functions. *Section 6.10: Port8*: *Figure 53 on page 176* updated - unidirectional arrows on XPWM alternate functions. Section *Section 15.1: Selection among user-code, standard or alternate bootstrap on page 302*: Decoding of P0L.5 = 1, P0L.4 = 0 description updated. *Section 22: Real time clock on page 464*: Introduction updated clarifying the behavior of "cyclic time based interrupt" bullet. *Section 22: Real time clock on page 464*: Description of bit RTCAEN of RTCCON register updated. *Section 22: Real time clock on page 464*: Notes referring to RTCCON register description updated. *Section 23: System reset on page 472*: Some occurences of EA corrected in $\overline{EA}$. *Section 23.2: Asynchronous reset on page 472*: Note in "Power-On Reset" paragraph updated (missing word added). *Section 26: Register set on page 515*: Paragraph titles and table headers updated for coherency throughout. *Section 26: Register set on page 515*: Reset value of IDPROG register corrected. *Section 26: Register set on page 515*: Name of all I^2C registers corrected adding prefix "I2C".
16-Feb-2007	0.7	Added *Section 4.2.6: The 40-bit signed accumulator register on page 87*. *Table 44 on page 300*: Added cross references to footnotes 1 and 2 *Section 16.5.1 on page 337*: Added cross reference to *Double register compare mode* *Section 16.5.2 on page 338*: Added cross reference to *Double register compare mode* in first note *Table 64 on page 495*: Added cross reference to footnote 1 Document format updated
6-Mar-2007	1	Make changes to tables & figures to comply with corporate template.
07-May-2013	2	Updated *Chapter 20: I2C interface*

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT AUTHORIZED FOR USE IN WEAPONS. NOR ARE ST PRODUCTS DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

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

www.st.com