RM0001
Reference manual

STR73x ARMTDMI® microcontroller family

Introduction

This Reference Manual provides complete information for application developers on how to use the STR73x Microcontroller memory and peripherals.

For Ordering Information, Mechanical and Electrical Device Characteristics please refer to the STR73x Datasheet.

For information on programming, erasing and protection of the internal Flash memory please refer to the STR7 Flash Programming Reference Manual

For information on the ARM7TDMI® core please refer to the ARM7TDMI Technical Reference Manual.

Contents

1 **Related documents 14**

2 **Memory 15**

2.1 Memory organization 15

2.1.1 Memory map 16

2.1.2 Register base addresses 17

2.1.3 APB memory map 17

2.1.4 Boot memory 21

2.1.5 RAM 21

2.1.6 Flash 22

2.1.7 Flash power down mode 22

2.2 Boot configuration 23

2.2.1 SystemMemory boot mode 23

3 **Power, reset and clocks 24**

3.1 Power supply 24

3.2 Reset 24

3.2.1 Reset pin timing 26

3.2.2 LVD reset 26

3.2.3 Power-on 27

3.2.4 Voltage drop 28

3.3 Clocks 29

3.3.1 RC oscillator 30

3.4 Low power modes 30

3.4.1 Slow mode 30

3.4.2 WFI mode 30

3.4.3 LPWFI mode 31

3.4.4 Halt mode 32

3.4.5 Stop mode 32

3.5 Clock monitor unit (CMU) 32

3.5.1 Introduction 32

3.5.2 Register write protection 33

3.5.3 Clock source selection 33

3.5.4 Oscillator frequency monitoring 33

3.5.5 MCLK frequency monitoring 34
3.5.6 Clock frequency measurement 34
3.5.7 RC oscillator control 34
3.5.8 Limitations 35
3.5.9 Register description 35
3.5.10 RC oscillator control register (CMU_RCCTL) 35
3.5.11 Frequency display register (CMU_FDISP) 36
3.5.12 Frequency reference high register (CMU_FRH) 36
3.5.13 Frequency reference low register (CMU_FRL) 36
3.5.14 Control register (CMU_CTRL) 37
3.5.15 Status register (CMU_STAT) 38
3.5.16 Interrupt status register (CMU_IS) 39
3.5.17 Interrupt mask register (CMU_IM) 40
3.5.18 End of count value register (CMU_EOCV) 40
3.5.19 Write enable register (CMU_WE) 41
3.5.20 CMU register map 41
3.6 Power, reset and clock control unit (PRCCU) 42
3.6.1 Overview 42
3.6.2 PLL clock multiplier programming 42
3.6.3 Peripheral clocks 44
3.6.4 RT clock (fEXT) 44
3.6.5 Clock configuration reset state 44
3.6.6 Interrupt generation 44
3.6.7 Register description 45
3.6.8 Clock control register (PRCCU_ CCR) 45
3.6.9 Voltage regulator control register (PRCCU_VRCTR) 46
3.6.10 Clock flag register (PRCCU_CFR) 47
3.6.11 PLL configuration register (PRCCU_PLLCR) 49
3.6.12 System mode register (PRCCU_SMR) 50
3.6.13 Real time clock programming register (PRCCU_RTCPR) 51
3.6.14 PRCCU register map 52

4 Configuration registers (CFG) 53

4.1 System configuration registers 53
4.1.1 Configuration register 0 (CFG_R0) 53
4.1.2 Configuration register 1 (CFG_R1) 54
4.1.3 Device identification register (CFG_DIDR) 55

4.2 External interrupt request configuration registers 55
4.2.1 External interrupt trigger event register 0 (CFG_EITE0) 55
4.2.2 External interrupt trigger event register 1 (CFG_EITE1) 56
4.2.3 External interrupt trigger event register (CFG_EITE2) 56
4.3 Peripheral clock management registers 56
4.3.1 Clock management in user mode 56
4.3.2 Peripheral clock gating register 0 (CFG_PCGR0) 57
4.3.3 Peripheral clock gating register 1 (CFG_PCGR1) 59
4.3.4 Peripheral clock gating register B0 (CFG_PCGRB0) 60
4.3.5 Peripheral clock gating register B1 (CFG_PCGRB1) 61
4.3.6 TIM external clock select register (CFG_TIMSR) 61
4.3.7 Clock management in emulation mode 61
4.3.8 Peripheral emulation clock gating register 0 (CFG_PECGR0) 62
4.3.9 Peripheral emulation clock gating register 1 (CFG_PECGR1) 62
4.4 BSPI and UART management in emulation mode 63
4.4.1 Emulation serial protection register (CFG_ESPR) 63
4.5 CFG register map 64

5 Clock tree map 65

6 I/O ports 67
6.1 Functional description 67
6.1.1 General purpose I/O (GPIO) 68
6.1.2 Alternate function I/O (AF) 68
6.1.3 Input configuration 69
6.1.4 Input pull up/pull down configuration 69
6.1.5 Output configuration 70
6.1.6 Alternate Function configuration 70
6.1.7 High impedance-analog input configuration 71
6.2 Register description 72
6.2.1 I/O Port register map 73

7 Interrupts 74
7.1 Enhanced interrupt controller (EIC) 75
7.1.1 IRQ interrupt vector table 76
7.1.2 FIQ interrupt vector table 78
7.1.3 IRQ interrupt structure 78

7.1.4 Priority decoder . . . 80
7.1.5 Finite state machine . . . 81
7.1.6 Stack . . . 81
7.1.7 EIC interrupt vectoring . . . 82
7.1.8 EIC IRQ notes . . . 83
7.2 FIQ mechanism . . . 85
7.3 Register programming . . . 85
7.4 Application note . . . 86
7.4.1 Avoiding LR_sys and r5 registers content loss . . . 87
7.4.2 Hints about subroutines used inside ISRs . . . 87
7.5 Interrupt latency . . . 87
7.6 Register description . . . 89
7.6.1 Interrupt control register (EIC_ICR) . . . 89
7.6.2 Current interrupt channel register (EIC_CICR) . . . 90
7.6.3 Current interrupt priority register (EIC_CIPR) . . . 91
7.6.4 Fast interrupt enable register (EIC_FIER) . . . 92
7.6.5 Fast interrupt pending register (EIC_FIPR) . . . 92
7.6.6 Interrupt vector register (EIC_IVR) . . . 93
7.6.7 Fast interrupt register (EIC_FIR) . . . 94
7.6.8 Interrupt enable register 0 (EIC_IER0) . . . 94
7.6.9 Interrupt enable register 1 (EIC_IER1) . . . 95
7.6.10 Interrupt pending register 0 (EIC_IPR0) . . . 96
7.6.11 Interrupt pending register 1 (EIC_IPR1) . . . 97
7.6.12 Source interrupt registers - channel n (EIC_SIRn) . . . 98
7.7 EIC register map . . . 99
7.8 External interrupt pins INT[15:0] . . . 102
7.8.1 Edge-triggered external interrupts . . . 102
7.8.2 Level-triggered external interrupts . . . 102
7.9 Wake-up interrupt unit (WIU) . . . 103
7.9.1 Features . . . 103
7.9.2 Functional description . . . 105
7.9.3 Interrupt mode . . . 105
7.9.4 Wake-up mode selection . . . 105
7.9.5 Stop mode entry conditions . . . 105
7.9.6 Programming considerations . . . 106
7.9.7 Procedure for entering/exiting Stop mode . . . 107

7.9.8 Simultaneous setting of pending bits . . . 107
7.9.9 Register description . . . 108
7.9.10 Wake-up control register (WIU_CTRL) . . . 108
7.9.11 Wake-up mask register (WIU_MR) . . . 109
7.9.12 Wake-up trigger register (WIU_TR) . . . 110
7.9.13 Wake-up software interrupt register (WIU_INTR) . . . 110
7.9.14 Wake-up pending register (WIU_PR) . . . 111
7.9.15 WIU register map . . . 111

8 DMA Controller (DMA) . . . 112
8.1 Introduction . . . 112
8.2 DMA controller priority . . . 112
8.3 DMA request mapping . . . 112
8.4 Functional description . . . 113
8.5 Register description . . . 116
8.6 DMA register map . . . 127

9 Native bus arbiter (ARB) . . . 129
9.1 Register description . . . 129
9.1.1 Time-Out Register (ARB_TOR) . . . 129
9.1.2 Priority Register (ARB_PRIOR) . . . 129
9.1.3 Control Register (ARB_CTLR) . . . 130
9.2 Native bus arbiter register map . . . 130

10 Wake-up timer (WUT) . . . 131
10.1 Introduction . . . 131
10.2 Main features . . . 131
10.3 Functional description . . . 131
10.3.1 Free-running timer mode . . . 132
10.4 Programming considerations . . . 132
10.5 Register Description . . . 133
10.5.1 Wake-up Timer Control Register (WUT_CR) . . . 133
10.5.2 Wake-up Timer Prescaler Register (WUT_PR) . . . 133
10.5.3 Wake-up Timer Pre-load Value Register (WUT_VR) . . . 134
10.5.4 Wake-up Timer Counter Register (WUT_CNT) . . . 134
10.5.5 Wake-up Timer Status Register (WUT_SR) . . . 134

10.5.6 Wake-up Timer Mask Register (WUT_MR) 135
10.6 WUT register map 135

11 Real time clock (RTC) 136
11.1 Introduction 136
11.2 Main features 136
11.3 Functional description 137
11.3.1 Overview 137
11.3.2 Free-running mode 138
11.3.3 RTC flag assertion 138
11.3.4 Configuration mode 139
11.4 Register description 140
11.4.1 RTC Control Register High (RTC_CRH) 140
11.4.2 RTC Control Register Low (RTC_CRL) 141
11.4.3 RTC Prescaler Load Register High (RTC_PRLH) 143
11.4.4 RTC Prescaler Load Register Low (RTC_PRLL) 143
11.4.5 RTC Prescaler Divider Register High (RTC_DIVH) 143
11.4.6 RTC Prescaler Divider Register Low (RTC_DIVL) 144
11.4.7 RTC Counter Register High (RTC_CNTH) 144
11.4.8 RTC Counter Register Low (RTC_CNTL) 144
11.4.9 RTC Alarm Register High (RTC_ALRH) 145
11.4.10 RTC alarm register Low (RTC_ALRL) 145
11.5 RTC register map 145

12 Watchdog timer (WDG) 147
12.1 Introduction 147
12.2 Main Features 147
12.3 Functional description 147
12.3.1 Free-running timer mode 147
12.3.2 Watchdog mode 148
12.4 Register description 148
12.4.1 WDG control register (WDG_CR) 149
12.4.2 WDG prescaler register (WDG_PR) 149
12.4.3 WDG preload value register (WDG_VR) 149
12.4.4 WDG counter register (WDG_CNT) 150
12.4.5 WDG status register (WDG_SR) 151

12.4.6 WDG mask register (WDG_MR) . . . 151
12.4.7 WDG key register (WDG_KR) . . . 151
12.5 WDG register map . . . 152

13 Timebase timer (TB) . . . 153
13.1 Main features . . . 153
13.2 Functional description . . . 153
13.2.1 Free-running timer mode . . . 154
13.3 Register description . . . 154
13.4 TB register map . . . 156

14 Timer (TIM) . . . 157
14.1 Introduction . . . 157
14.2 Main features . . . 157
14.3 Functional description . . . 158
14.3.1 Counter . . . 158
14.3.2 External clock . . . 159
14.3.3 Input capture . . . 161
14.3.4 Procedure . . . 161
14.3.5 Output compare . . . 162
14.3.6 Procedure . . . 163
14.3.7 Forced compare mode . . . 164
14.3.8 One pulse mode . . . 165
14.3.9 Procedure . . . 165
14.3.10 Pulse width modulation mode . . . 166
14.3.11 Procedure . . . 167
14.3.12 Pulse width modulation input . . . 169
14.3.13 Procedure . . . 169
14.4 Interrupt management . . . 171
14.4.1 Use of interrupt channels . . . 171
14.5 DMA function . . . 171
14.6 Register description . . . 171
14.6.1 Input Capture A Register (TIMn_ICAR) . . . 171
14.6.2 Input capture B register (TIMn_ICBR) . . . 172
14.6.3 Output compare A register (TIMn_OCAR) . . . 172
14.6.4 Output compare B register (TIMn_OCBR) . . . 172

14.6.5 Counter register (TIMn_CNTR) . . . 172
14.6.6 Control register 1 (TIMn_CR1) . . . 173
14.6.7 Control register 2 (TIMn_CR2) . . . 174
14.6.8 Status register (TIMn_SR) . . . 175
14.7 TIM register map . . . 176

15 Pulse width modulator (PWM) . . . 177
15.1 Introduction . . . 177
15.2 Main features . . . 177
15.3 Functional description . . . 177
15.3.1 PWM operating mode . . . 178
15.3.2 Formulas . . . 178
15.4 Register description . . . 180
15.4.1 Prescaler 0 register (PWMn_PRS0) . . . 180
15.4.2 Prescaler 1 register (PWMn_PRS1) . . . 180
15.4.3 PWM enable register (PWMn_PEN) . . . 180
15.4.4 PWM output polarity level selection (PWMn_PLS) . . . 181
15.4.5 PWM compare period interrupt (PWMn_CPI) . . . 181
15.4.6 PWM interrupt mask register (PWMn_IM) . . . 181
15.4.7 PWM output duty register (PWMn_DUT) . . . 182
15.4.8 PWM output period register (PWMn_PER) . . . 182
15.5 PWM register map . . . 182

16 Controller area network (CAN) . . . 183
16.1 Introduction . . . 183
16.2 Main features . . . 183
16.3 Block diagram . . . 183
16.4 Functional description . . . 184
16.4.1 Software initialization . . . 184
16.4.2 CAN message transfer . . . 185
16.4.3 Disabled automatic re-transmission mode . . . 185
16.4.4 Test mode . . . 186
16.5 Register description . . . 188
16.5.1 CAN interface reset state . . . 190
16.5.2 CAN protocol related registers . . . 190
16.5.3 Message interface register sets . . . 195

16.5.4 Message handler registers 204
16.6 CAN register map 208
16.7 CAN communications 209
16.7.1 Managing message objects 209
16.7.2 Message handler state machine 210
16.7.3 Configuring a transmit object 213
16.7.4 Updating a transmit object 213
16.7.5 Configuring a receive object 214
16.7.6 Handling received messages 214
16.7.7 Configuring a FIFO buffer 215
16.7.8 Receiving messages with FIFO buffers 215
16.7.9 Handling interrupts 216
16.7.10 Configuring the bit timing 217

17 I2C interface module (I2C) 226
17.1 Main features 226
17.2 General description 226
17.2.1 Mode selection 227
17.2.2 Communication flow 227
17.2.3 SDA/SCL line control 227
17.3 Functional description 228
17.3.1 Slave mode 229
17.3.2 Slave receiver 229
17.3.3 Slave transmitter 229
17.3.4 Closing slave communication 229
17.3.5 Error cases 230
17.3.6 How to release the SDA / SCL lines 230
17.3.7 Master mode 230
17.3.8 Start condition 230
17.3.9 Slave address transmission 230
17.3.10 Master receiver 231
17.3.11 Master transmitter 231
17.3.12 Error cases 231
17.4 Interrupts 233
17.5 Register description 234
17.5.1 I2C Control Register (I2Cn_CR) 234

17.5.2 I2C Status Register 1 (I2Cn_SR1) . . . 235
17.5.3 I2C Status Register 2 (I2Cn_SR2) . . . 237
17.5.4 I2C Clock Control Register (I2Cn_CCR) . . . 238
17.5.5 I2C Extended Clock Control Register (I2Cn_ECCR) . . . 239
17.5.6 I2C Own Address Register 1 (I2Cn_OAR1) . . . 239
17.5.7 I2C Own Address Register 2 (I2Cn_OAR2) . . . 239
17.5.8 I2C Data Register (I2Cn_DR) . . . 240
17.6 I2C register map . . . 241

18 Buffered SPI (BSPI) . . . 242
18.1 Main features . . . 242
18.2 Functional description . . . 242
18.2.1 BSPI pin description . . . 243
18.2.2 BSPI operation . . . 244
18.2.3 Transmit FIFO . . . 246
18.2.4 Receive FIFO . . . 246
18.2.5 Start-up status . . . 247
18.2.6 Clocking problems and clearing of the shift-register . . . 247
18.2.7 Interrupt control . . . 247
18.2.8 DMA interface . . . 247
18.3 Register description . . . 248
18.3.1 BSPI control/status register 1 (BSPIn_CSR1) . . . 248
18.3.2 BSPI control/status register 2 (BSPIn_CSR2) . . . 250
18.3.3 BSPI control/status register 3 (BSPIn_CSR3) . . . 252
18.3.4 BSPI master clock divider register (BSPIn_CLK) . . . 254
18.3.5 BSPI transmit register (BSPIn_TXR) . . . 254
18.3.6 BSPI receive register (BSPIn_RXR) . . . 255
18.4 BSPI register map . . . 255

19 UART . . . 256
19.1 Introduction . . . 256
19.2 **Main** features . . . 256
19.3 Functional description . . . 256
19.3.1 Transmission . . . 257
19.3.2 Reception . . . 258
19.3.3 Timeout mechanism . . . 259

19.3.4 Baud rate generation . . . 259
19.3.5 Interrupt control . . . 261
19.3.6 Using the UART interrupts when FIFOs are disabled . . . 261
19.3.7 Using the UART interrupts when FIFOs are enabled . . . 262
19.4 Register description . . . 262
19.4.1 UART baudrate register (UARTn_BR) . . . 262
19.4.2 UART TxBuffer register (UARTn_TxBUFR) . . . 263
19.4.3 UART RxBuffer register (UARTn_RxBUFR) . . . 263
19.4.4 UART control register (UARTn_CR) . . . 264
19.4.5 UART IntEnable register (UARTn_IER) . . . 265
19.4.6 UART status register (UARTn_SR) . . . 266
19.4.7 UART timeout register (UARTn_TOR) . . . 267
19.4.8 UART TxReset register (UARTn_TxRSTR) . . . 267
19.4.9 UART RxReset register (UARTn_RxRSTR) . . . 267
19.5 UART register map . . . 268

20 A/D converter (ADC) . . . 269
20.1 Main features . . . 269
20.2 Introduction . . . 269
20.3 Functional description . . . 270
20.3.1 Start of calibration . . . 270
20.3.2 Start of conversion . . . 271
20.3.3 Operating modes . . . 271
20.3.4 Input channel selection . . . 271
20.3.5 Analog clock prescaler . . . 271
20.3.6 Injected conversion chain . . . 273
20.3.7 Analog watchdogs . . . 273
20.3.8 DMA functionality . . . 274
20.3.9 Interrupts . . . 274
20.3.10 Power down mode . . . 274
20.3.11 Auto-clock-off mode . . . 275
20.4 Register description . . . 275
20.4.1 ADC Data Register (ADC_Dx) . . . 275
20.4.2 Control Logic Register 0 (ADC_CLR0) . . . 275
20.4.3 Control Logic Register 1 (ADC_CLR1) . . . 276
20.4.4 Control logic register 2 (ADC_CLR2) . . . 277

20.4.5 Control logic register 3 (ADC_CLR3) 277
20.4.6 Control logic register 4 (ADC_CLR4) 278
20.4.7 Threshold registers A (ADC_TRA0 ..3) 278
20.4.8 Threshold registers B (ADC_TRB0 ..3) 279
20.4.9 DMA channel enable register (ADC_DMAR) 279
20.4.10 DMA global enable register (ADC_DMAE) 279
20.4.11 Pending bit register (ADC_PBR) 280
20.4.12 Interrupt mask register (ADC_IMR) 281
20.5 ADC register map 283

21 APB bridge 284
21.1 Register description 284
21.2 APB register map 287

22 JTAG interface 288
22.1 Pins and reset status 288
22.1.1 JTAG ID code 288

23 Revision history 289

1 Related documents

Available from www.arm.com:

ARM7TDMI Technical Reference Manual

Available from www.st.com:

STR73xF Datasheet

STR7 Flash Programming Reference Manual

2 Memory

2.1 Memory organization

Program memory, data memory, registers and I/O ports are organized within the same linear address space of 4 GBytes.

The bytes are treated in memory as being in Little Endian format. The lowest numbered byte in a word is considered the word's least significant byte and the highest numbered byte the most significant.

Figure 1 on page 16 shows the STR73x Memory Map in Boot mode 1 after reset. For the detailed mapping of peripheral registers, please refer to the related chapters.

The addressable memory space is divided into 8 main blocks, selected by the three most significant bits of the memory address bus A[31:0]:

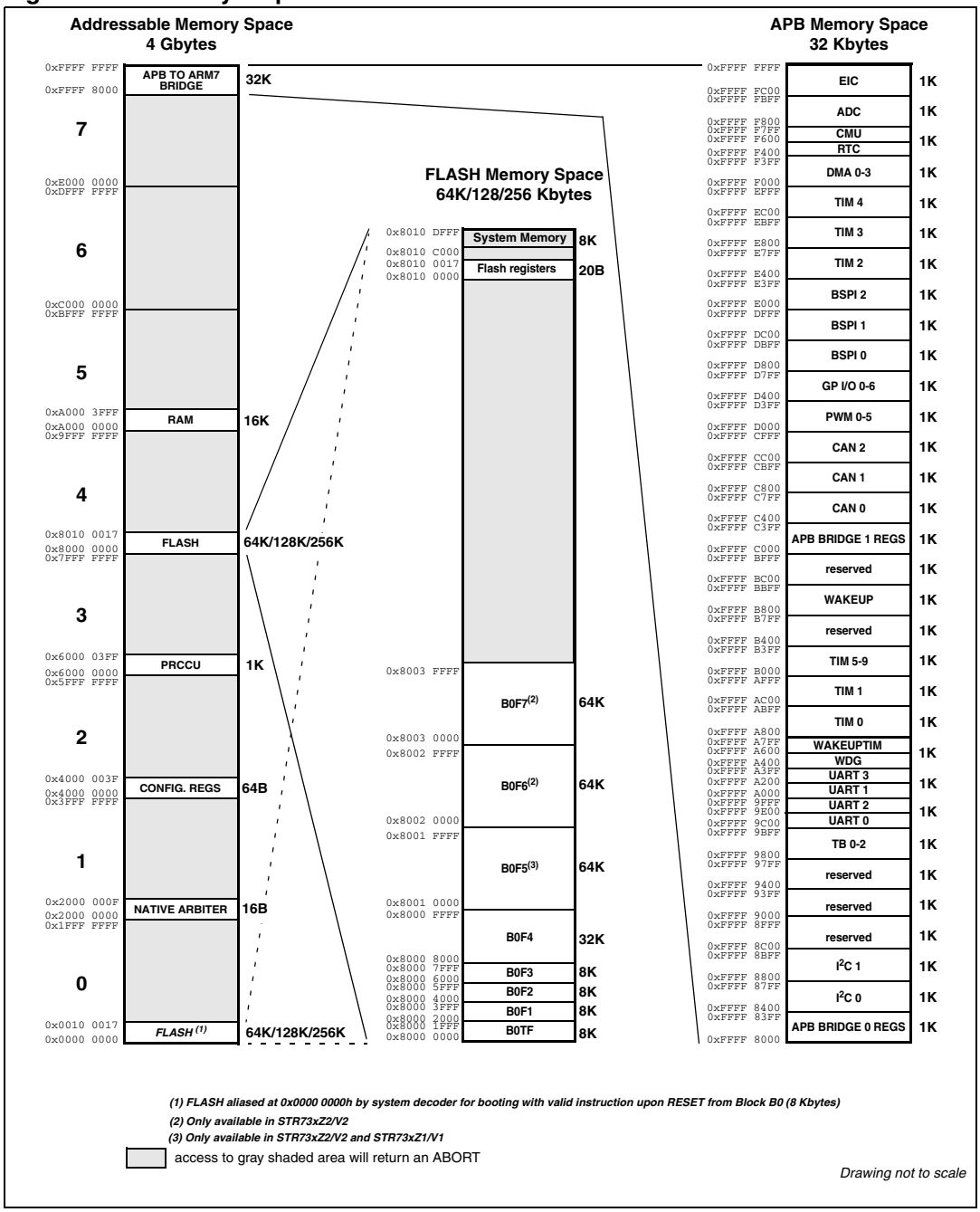
- 000 = Boot Memory
- 001 = Reserved
- 010 = System Configuration Registers (CFG)
- 011 = Reset and Clock Control Unit (PRCCU)
- 100 = Flash Memory
- 101 = RAM Memory
- 110 = Reserved
- 111 = APB Bridges including Enhanced Interrupt Controller

STR73x devices have no external memory interface. All memory spaces that are not allocated to on-chip memories and peripherals are considered "reserved".

Most reserved memory spaces (gray shaded areas in the *Figure 1 on page 16*) are protected from access by user code. When an access this memory space is attempted, an ABORT signal is generated. Depending on the type of access, the ARM processor will enter "prefetch abort" state (Exception vector 0x0000_000C) or "data abort" state (Exception vector 0x0000_0010). It is up to the application software to manage these abort exceptions.

2.1.1 Memory map

Figure 1. Memory map



2.1.2 Register base addresses

Table 1. Main block register base addresses

STR73x Main Blocks	Base Address	Block Register Map
Configuration Registers (CFG)	0x4000 0000	*Section 4.5*
Power, Reset and Clock Control Unit (PRCCU)	0x6000 0000	*Section 3.6.14*
Flash	0x8000 0000	See STR7xx Family Flash Programming Reference Manual

2.1.3 APB memory map

Table 2. APB memory map

Sub page	SubPage boundary addresses	Peripheral	Peripheral boundary addresses	Bus access width	Register map
0	0xFFFF 8000 0xFFFF 83FF	APB Bridge 0 Registers	0xFFFF 8000 0xFFFF 800F	32-bit	*Section 21.2*
1	0x FFFF 8400 0x FFFF 87FF	I^2C 0	0x FFFF 8400 0xFFFF 841F	8 bit	*Section 17.6*
2	0x FFFF 8800 0x FFFF 8BFF	I^2C 1	0x FFFF 8800 0xFFFF 881F	8 bit	
3	0x FFFF 8C00 0x FFFF 8FFF	reserved			
4	0x FFFF 9000 0x FFFF 93FF	reserved			
5	0x FFFF 9400 0x FFFF 97FF	reserved			
6	0x FFFF 9800 0x FFFF 9BFF	Timebase (TB) Timer 0	0x FFFF 9800 0x FFFF 9817	16 bit	*Section 13.4*
		Timebase (TB) Timer 1	0x FFFF 9900 0x FFFF 9917	16 bit	
		Timebase (TB) Timer 2	0x FFFF 9A00 0x FFFF 9A17	16 bit	

Table 2. APB memory map (continued)

Sub page	SubPage boundary addresses	Peripheral	Peripheral boundary addresses	Bus access width	Register map
7	0x FFFF 9C00 0x FFFF 9FFF	UART 0	0x FFFF 9C00 0x FFFF 9C27	16 bit	Section 19.5
		UART 2	0x FFFF 9E00 0x FFFF 9FFF	16 bit	
8	0x FFFF A000 0x FFFF A3FF	UART 1	0x FFFF A000 0x FFFF A027	16 bit	
		UART 3	0x FFFF A200 0x FFFF A3FF	16 bit	
9	0x FFFF A400 0x FFFF A7FF	Watchdog (WDG)	0x FFFF A400 0x FFFF A41B	16 bit	Section 12.5
		Wake-up Timer (WUT)	0x FFFF A600 0x FFFF A617	16 bit	Section 10.6
10	0x FFFF A800 0x FFFF ABFF	TIM 0	0x FFFF A800 0x FFFF A81F	16 bit	Section 14.7
11	0x FFFF AC00 0x FFFF AFFF	TIM 1	0x FFFF AC00 0x FFFF AC1F	16 bit	
12	0x FFFF B000 0x FFFF B3FF	TIM 5	0x FFFF B000 0x FFFF B01F	16 bit	
		TIM 6	0x FFFF B080 0x FFFF B09F	16 bit	
		TIM 7	0x FFFF B100 0x FFFF B11F	16 bit	
		TIM 8	0x FFFF B180 0x FFFF B19F	16 bit	
		TIM 9	0x FFFF B200 0x FFFF B21F	16 bit	
13	0x FFFF B400 0x FFFF B7FF	reserved			
14	0x FFFF B800 0x FFFF BBFF	Wake-up/Interrupt Unit (WIU)	0x FFFF B800 0x FFFF B813	32 bit	Section 7.9.15
15	0x FFFF BC00 0x FFFF BFFF	reserved			

Table 2. APB memory map (continued)

Sub page	SubPage boundary addresses	Peripheral	Peripheral boundary addresses	Bus access width	Register map
16	0x FFFF C000 0x FFFF C3FF	APB Bridge 1 Registers	0xFFFF C000 0xFFFF C00F	32-bit	*Section 21.2*
17	0x FFFF C400 0x FFFF C7FF	CAN 0	0x FFFF C400 0x FFFF C57F	16 bit	*Section 16.6*
18	0x FFFF C800 0x FFFF CBFF	CAN 1	0x FFFF C800 0x FFFF C97F	16 bit	
19	0x FFFF CC00 0x FFFF CFFF	CAN 2	0x FFFF CC00 0x FFFF CFFF	16 bit	
20	0x FFFF D000 0x FFFF D3FF	PWM 0	0x FFFF D000 0x FFFF D023	16 bit	*Section*
		PWM 1	0x FFFF D040 0x FFFF D063	16 bit	
		PWM 2	0x FFFF D080 0x FFFF D0A3	16 bit	
		PWM 3	0x FFFF D0C0 0x FFFF D0E3	16 bit	
		PWM 4	0x FFFF D100 0x FFFF D123	16 bit	
		PWM 5	0x FFFF D140 0x FFFF D163	16 bit	

Table 2. APB memory map (continued)

Sub page	SubPage boundary addresses	Peripheral	Peripheral boundary addresses	Bus access width	Register map
21	0x FFFF D400 0x FFFF D7FF	GP I/O - Port 0	0x FFFF D400	16 bit	Section 6.2.1
			0x FFFF D40F		
		GP I/O - Port 1	0x FFFF D410	16 bit	
			0x FFFF D41F		
		GP I/O - Port 2	0x FFFF D420	16 bit	
			0x FFFF D42F		
		GP I/O - Port 3	0x FFFF D430	16 bit	
			0x FFFF D43F		
		GP I/O - Port 4	0x FFFF D440	16 bit	
			0x FFFF D44F		
		GP I/O - Port 5	0x FFFF D450	16 bit	
			0x FFFF D45F		
		GP I/O - Port 6	0x FFFF D460	16 bit	
			0x FFFF D46F		
22	0x FFFF D800	BSPI 0	0x FFFF D800	16 bit	Section 18.4
	0x FFFF DBFF		0x FFFF D817		
23	0x FFFF DC00	BSPI 1	0x FFFF DC00	16 bit	
	0x FFFF DFFF		0x FFFF DC17		
24	0x FFFF E000	BSPI 2	0x FFFF E000	16 bit	
	0x FFFF E3FF		0x FFFF E017		
25	0x FFFF E400	TIM 2	0x FFFF E400	16 bit	Section 14.7
	0x FFFF E7FF		0x FFFF E41F		
26	0x FFFF E800	TIM 3	0x FFFF E800	16 bit	
	0x FFFF EBFF		0x FFFF E81F		
27	0x FFFF EC00	TIM 4	0x FFFF EC00	16 bit	
	0x FFFF EFFF		0x FFFF EC1F		

Table 2. APB memory map (continued)

Sub page	SubPage boundary addresses	Peripheral	Peripheral boundary addresses	Bus access width	Register map
28	0x FFFF F000 0x FFFF F3FF	DMA 0	0x FFFF F000 0x FFFF F0FB	16 bit	*Section 8.6*
		DMA 1	0x FFFF F100 0x FFFF F1FB	16 bit	
		DMA 2	0x FFFF F200 0x FFFF F2FB	16 bit	
		DMA 3	0x FFFF F300 0x FFFF F3FB	16 bit	
29	0x FFFF F400 0x FFFF F7FF	Real-time Clock (RTC)	0x FFFF F400 0x FFFF F427	16 bit	*Section 11.5*
		Clock Monitor Unit (CMU)	0x FFFF F600 0x FFFF F61F	16 bit	*Section 3.5.20*
30	0x FFFF F800 0x FFFF FBFF	Analog/Digital Converter (ADC)	0x FFFF F800 0x FFFF F94F	16 bit	*Section 20.5*
31	0x FFFF FC00 0x FFFF_FFFF	Enhanced Interrupt Controller (EIC)	0x FFFF FC00 0x FFFF FD5F	32 bit	*Section 7.7*

2.1.4 Boot memory

The boot mode is selected by the M0 and M1 pins on exit from Reset

- **User Boot Mode 1**: In this mode, Flash sector B0F0 is mapped in both Block 010 and Block 000 of the memory map. The system boots from block 0, segment 0 of Flash (normal operation)
- **User Boot Mode 2**: This mode has the same mapping as User Boot mode 1 except Flash sector B0F1 is reserved, any attempt to access address range 0x8000 2000 to 0x8000 3FFF will generate an ABORT.
- **SystemMemory Boot mode**: This mode has the same mapping as User Flash boot mode 1, except that the SystemMemory flash sector is accessible in address range 0x8010 C000 to 0x8010 DFFF and is aliased in Block 0 This allows the system to boot from SystemMemory (for initial Flash Programming).

2.1.5 RAM

The STR73x features 16 KBytes of static RAM. It can be accessed as bytes, half-words (16 bits) or full words (32 bits). The RAM start address is 0xA000 0000.

You can remap the RAM on-the-fly to Block 000, using the REMAP bit in the CFG_R0 register.

In REMAP mode the RAM start address is mapped both at 0x0000 0000h and at 0xA000 0000h.

This is particularly useful for managing interrupt vectors and routines, you can copy them to RAM, modify and access them even when Flash is not available (i.e. during Flash programming or erasing).

2.1.6 Flash

The Flash Module consists of a single bank (Bank 0) and is organized in sectors as shown in *Table 3*.

Table 3. Flash module organisation

Bank	Sector	Addresses	Size (bytes)
Bank 0 256 Kbytes Program Memory + 8K SystemMemory	Bank 0 Flash Sector 0 (B0F0)	0x00 0000 - 0x00 1FFF	8K
	Bank 0 Flash Sector 1 (B0F1)	0x00 2000 - 0x00 3FFF	8K
	Bank 0 Flash Sector 2 (B0F2)	0x00 4000 - 0x00 5FFF	8K
	Bank 0 Flash Sector 3 (B0F3)	0x00 6000 - 0x00 7FFF	8K
	Bank 0 Flash Sector 4 (B0F4)	0x00 8000 - 0x00 FFFF	32K
	Bank 0 Flash Sector 5 (B0F5)	0x01 0000 - 0x01 FFFF	64K(1)
	Bank 0 Flash Sector 6 (B0F6)	0x02 0000 - 0x02 FFFF	64K(2)
	Bank 0 Flash Sector7 (B0F7)	0x03 0000 - 0x03 FFFF	64K(2)
	Bank 0 Flash SystemMemory	0x10 C000 - 0x10 DFFF	8K
Flash Control Registers	Flash Control/Data Registers	0x10 0000 - 0x0010 0017	24
	Flash Protection Registers	0x10 DFB0 - 0x0010 DFBC	12

1. Not available in 64K versions.
2. Not available in 64K and 128K versions.

Sectors B0F0-B0F7 can be used as Boot sectors; they can be write protected against unwanted write operations.

Flash memory can be protected against different types of unwanted access (read/write/erase).

You can program Flash memory using In-Circuit Programming and In-Application programming. Refer to the STR7xx Flash Programming Reference Manual .

2.1.7 Flash power down mode

Depending on your application requirements, you can optionally power down the Flash in LPWFI mode (See *Section 3.4.3*). Otherwise, in LPWFI mode, the Flash automatically reduces its power consumption and can be read immediately after wake-up.

If you need even lower power consumption, you can put the Flash in Power-Down Mode. You do this by configuring the PWD bit in the FLASH_CR0 register. The consumption is drastically reduced, but after wake-up from LPWFI mode, a delay (t_{PD}) is inserted automatically to ensure the Flash is operational before the CPU restarts.

2.2 Boot configuration

The following sections describe the possible device Boot modes. In the STR73x, up to three different modes are available and can be enabled by means of two dedicated Input pins M1 and M0, as shown in *Table 4*.

Table 4. Boot modes

M1	M0	Boot Mode	Memory Mapping	Note
0	0	User Boot mode 1	FLASH sector B0F0 mapped at 0h	All FLASH sectors acessible except SystemMemory sector
0	1	User Boot mode 2	FLASH sector B0F0 mapped at 0h	FLASH B0F1 sector and SystemMemory sector not accessible
1	0	SystemMemory	SystemMemory mapped at 0h	-
1	1	Reserved	-	-

2.2.1 SystemMemory boot mode

SystemMemory Boot Mode is normally used when the FLASH is to be programmed for the first time. In this case the system boot is performed from SystemMemory sector. This mode allows to initialize the FLASH programming via a serial interface.

The SystemMemory code then loads a FLASH programming code (called “loader”) into internal RAM via one of the serial interfaces (JTAG, UART0, CAN0, depending on the user environment).

3 Power, reset and clocks

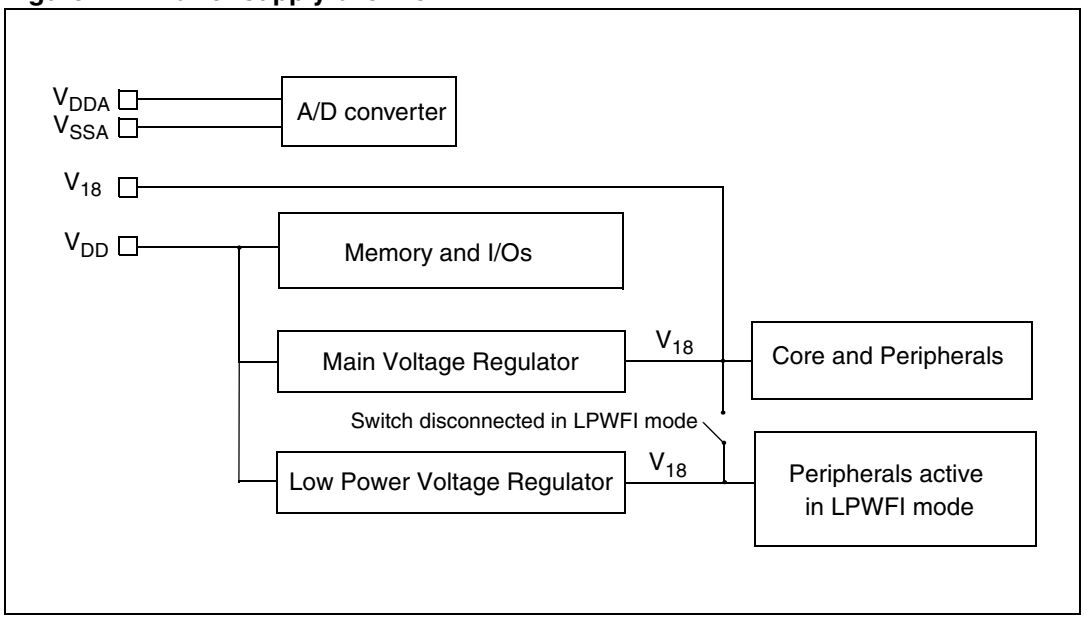
3.1 Power supply

The chip is powered by an external 5V supply: V_{DD}. All I/Os are 5V-capable. An internal Voltage Regulator generates the supply voltage for core logic (V_{18} =1.8 V). The V_{18} pin must be connected to a 100nF external stabilization capacitor.

In LPWFI mode the main voltage regulator can be powered off. In this mode, the low power regulator takes over and supplies power to the on-chip peripherals selected by the configuration registers.

The V_{DDA} and V_{SSA} pins supply the reference voltages for the A/D Converter.

Figure 2. Power supply overview



3.2 Reset

The Reset Manager resets the MCU when one of the following events occurs:

- An external reset, initiated by a low level on the $\overline{\text{RSTIN}}$ pin
- A software reset, forced by setting the HALT bit in the PRCCU_SMR register (when enabled by the SRESEN bit in the PRCCU_CCR register)
- A Watchdog end of count condition
- A voltage drop below the LVD threshold on V_{18}.

The event causing the last reset is flagged in the PRCCU_CFR register: the corresponding bit is set. An external reset will leave all these bits reset.

A reset overrides all other conditions and forces the system into reset state. During the reset phase, the internal registers are set to their reset values, and the I/O pin are configured in their reset state.

During the reset phase, the CMU automatically selects the RC oscillator clock (running at 2 MHz) as input clock to the PRCCU. *Figure 3 on page 25* shows a timing diagram of the reset phase. When the $\overline{\text{RSTIN}}$ pin goes high again, 404 RC-Oscillator clock cycles (CKOSC) plus 7 CLOCK2 cycles are counted before exiting reset state (plus eventually one CKOSC period, depending on the delay between the rising edge of the $\overline{\text{RSTIN}}$ pin and the first rising edge of CKOSC). It corresponds to about 209 us at 2 MHz RC-Oscillator frequency.

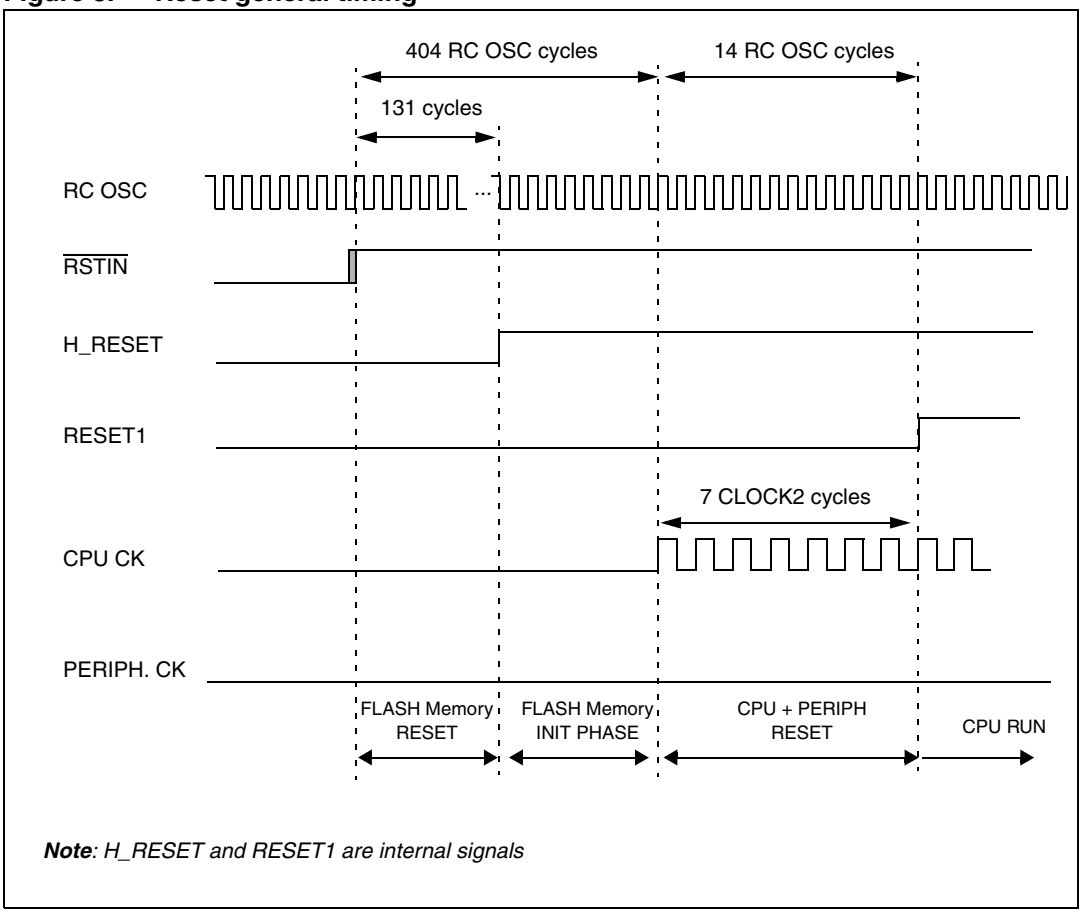
At the end of the Reset phase, the Program Counter is loaded with the value 0000 0000h and executes the instruction located at address 0000 0000h.

As shown in *Figure 3: Reset general timing on page 25*, after the $\overline{\text{RSTIN}}$ pin is released, 131 RC-Oscillator clock pulses are counted before the internal H_RESET signal is deasserted HIGH. The H_RESET signal is used to keep the Flash memory controller in reset state. When it goes high, the Flash memory controller starts to initialize the Flash. Then 418 RC-Oscillator clock pulses (404+14) after the $\overline{\text{RSTIN}}$ rising edge, the RESET1 signal is deasserted high, the clock signal to CPU is produced (CPU is in run mode) while clock signals to peripherals are still stretched low.

The peripherals are kept in reset state until their clocks are enabled by software. *Figure 4: Peripheral reset timings on page 26* shows the timing of this part of the reset phase.

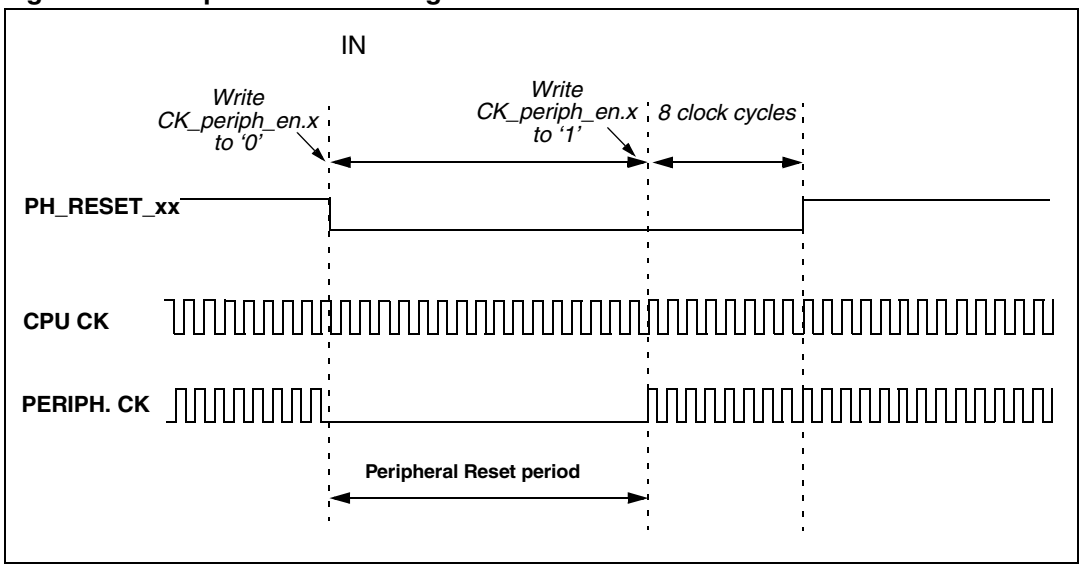
The peripheral are enabled individually by writing in the CFG_PCGR0 and CFG_PCGR1 registers (see *Section 4.3.1 on page 56*).

Figure 3. Reset general timing



***Note**: H_RESET and RESET1 are internal signals*

Figure 4. Peripheral reset timings

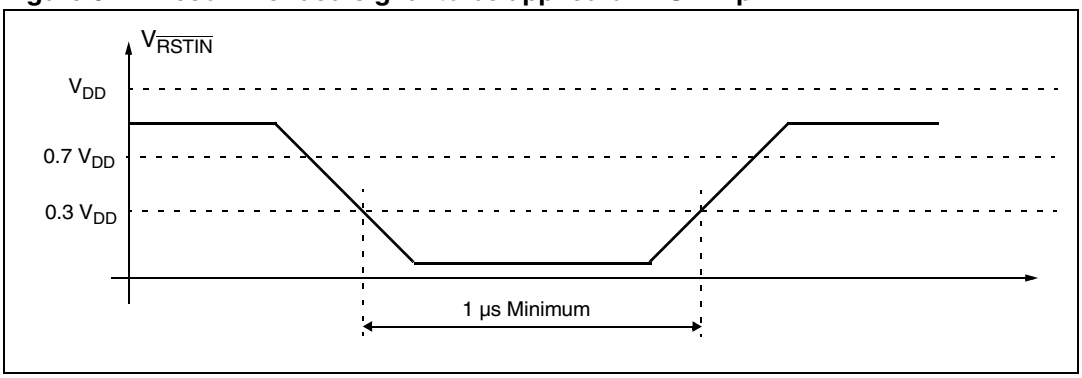


3.2.1 Reset pin timing

To improve the noise immunity of the device, the RESET input pin ($\overline{\text{RSTIN}}$) has a Schmitt trigger input circuit with hysteresis. Spurious RESET events are masked by an analog filter which guarantees that any glitches (single pulse and burst) on the $\overline{\text{RSTIN}}$ pin shorter than 100 ns are not recognized by the system as valid RESET pulses.

A valid pulse on $\overline{\text{RSTIN}}$ should have a duration of at least 1 υs to be sure that it is properly latched by system. This means that all the pulses longer than 100 ns and shorter than 1 υs can have an unpredictable effect on the device: they can either be recognized as valid or filtered.

Figure 5. Recommended signal to be applied on $\overline{\text{RSTIN}}$ pin



3.2.2 LVD reset

The internal LVDs (Low Voltage Detectors) guarantee the safe behavior of the device in the event of a drop in the internal voltage. They hold the device in reset while the voltage is below a specified threshold.

There are two separate LVD circuits in STR73x device: the first is embedded in the Voltage Regulator logic, the second one in the Flash module. They work at different thresholds: in

particular, the Voltage Regulator LVD has a higher threshold than the Flash LVD. Both work on internal V_{18}; no internal LVD circuit is provided on V_{DD}.

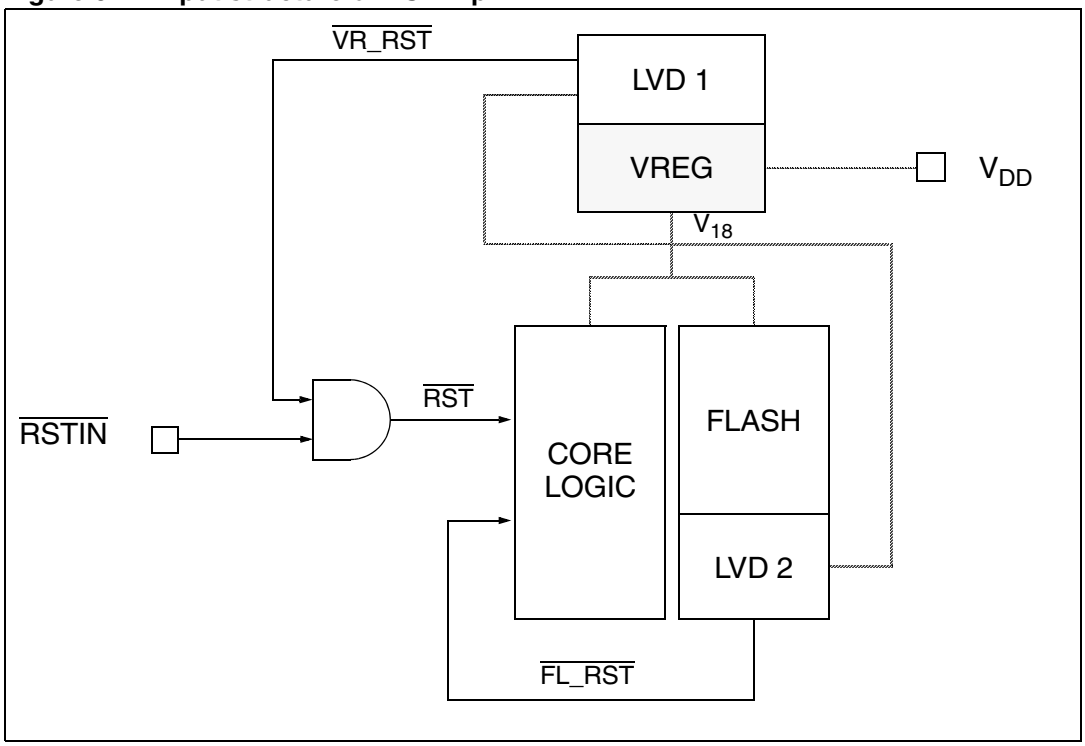
The following sections describe the behavior of the LVD circuits in two different situations: at Power-on and during a voltage drop on V_{18}. In both cases the LVD_INT bit in the PRCCU CLK_FLAG register status will flag the occurrence of an LVD Reset.

3.2.3 Power-on

At power-on, the $\overline{\text{RSTIN}}$ pin must be held low until V_{DD} is stable, to guarantee proper system initialization.

When V_{DD} is still too low to allow the internal Voltage Regulator to provide a proper V_{18} to the system, the LVD of the Voltage Regulator (monitoring V_{18} level) keeps the STR73x under Reset, injecting an internal reset through the input section of the $\overline{\text{RSTIN}}$ pin. The input Schmitt Trigger structure of the $\overline{\text{RSTIN}}$ pin is designed in such a way that, during this transition period, even though $\overline{\text{RSTIN}}$ pin is not properly driven (for example when V_{DD} is so low that the external regulator is not able to provide a well defined reset signal to the STR73x device), it internally forces a level '0' to the core logic, independent from the logic level externally present on the $\overline{\text{RSTIN}}$ pin (see next *Figure 6* for pin schematic, and *Figure 7* for wave diagrams).

Figure 6. Input structure of $\overline{\text{RSTIN}}$ pin



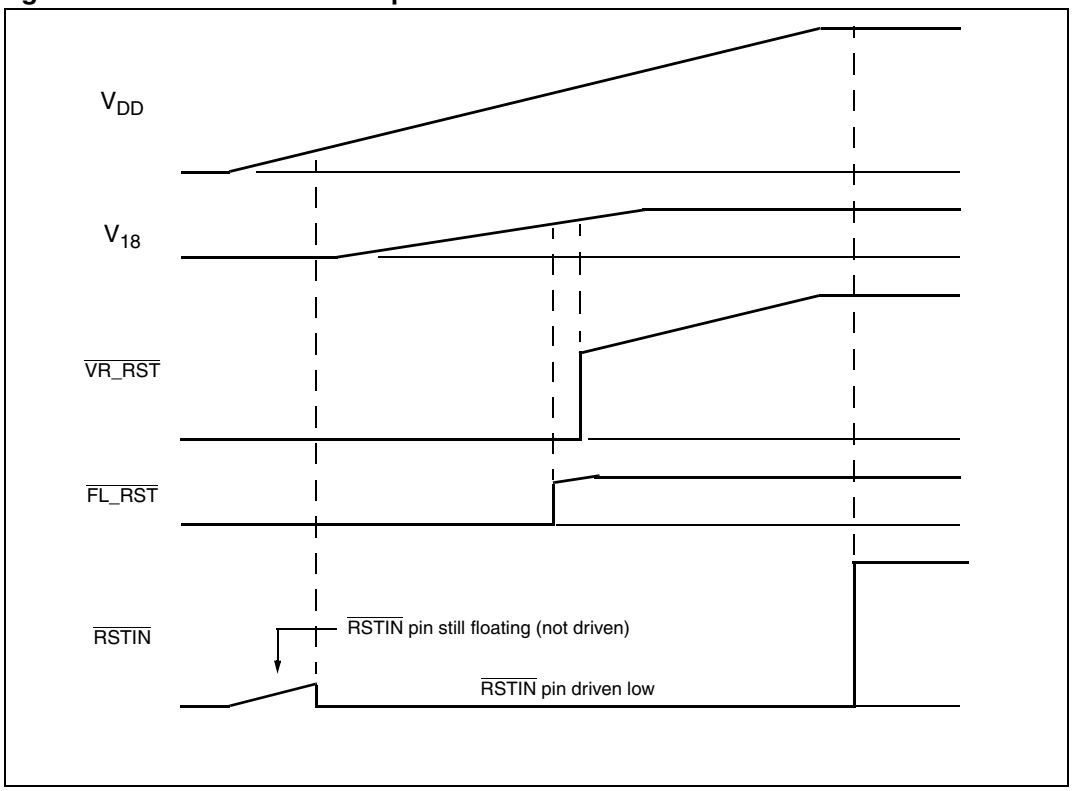
In this way the system sees a power-on reset and the flag register (PRCCU_CFR) is configured accordingly (all Reset flags cleared).

As soon as the internal Voltage Regulator is able to provide a sufficiently high voltage level to the core logic (with V_{DD} still ramping up, V_{18} reaches a value able to guarantee a proper power supply to all the internal logic circuitry), the LVD stops keeping the system under

reset. The embedded Flash module is ready to start working properly, since its embedded LVD has a lower threshold than the VREG one.

In next *Figure 7* a timing diagram of a typical power-on sequence is shown.

Figure 7. Power-on event sequence



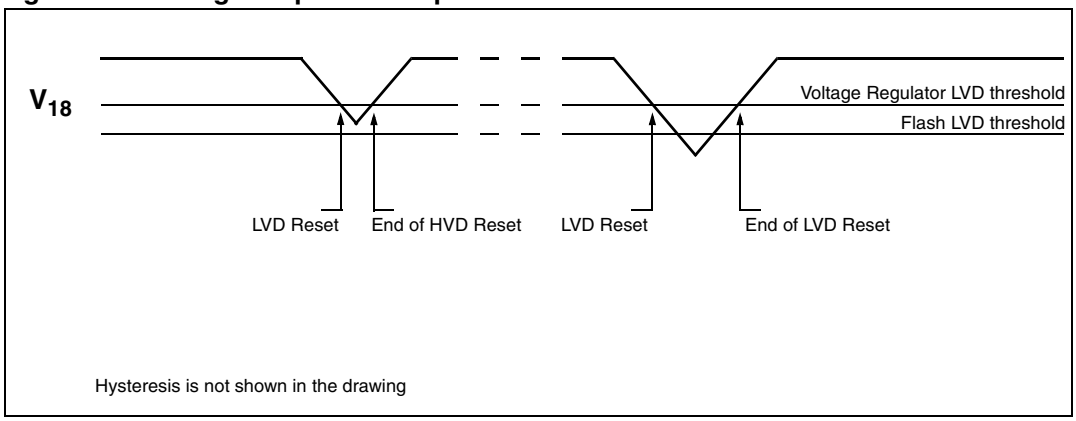
3.2.4 Voltage drop

When a temporary power drop on V_{18} occurs (below the Voltage Regulator LVD threshold), a system reset is generated. This Reset event is flagged in the PRCCU_CFR register. When the voltage comes back above the threshold, the system restarts with the usual reset exit sequence, and the software initialization routine can recognize the occurrence of a voltage drop by checking the status of the PRCCU_CFR register.

Next *Figure 8* shows how the system works during a typical voltage drop on V_{18}.

Note: Both LVD circuits has a typical hysteresis of about 100mV around the threshold value to guarantee the proper noise margin.

Figure 8. Voltage drop event sequence



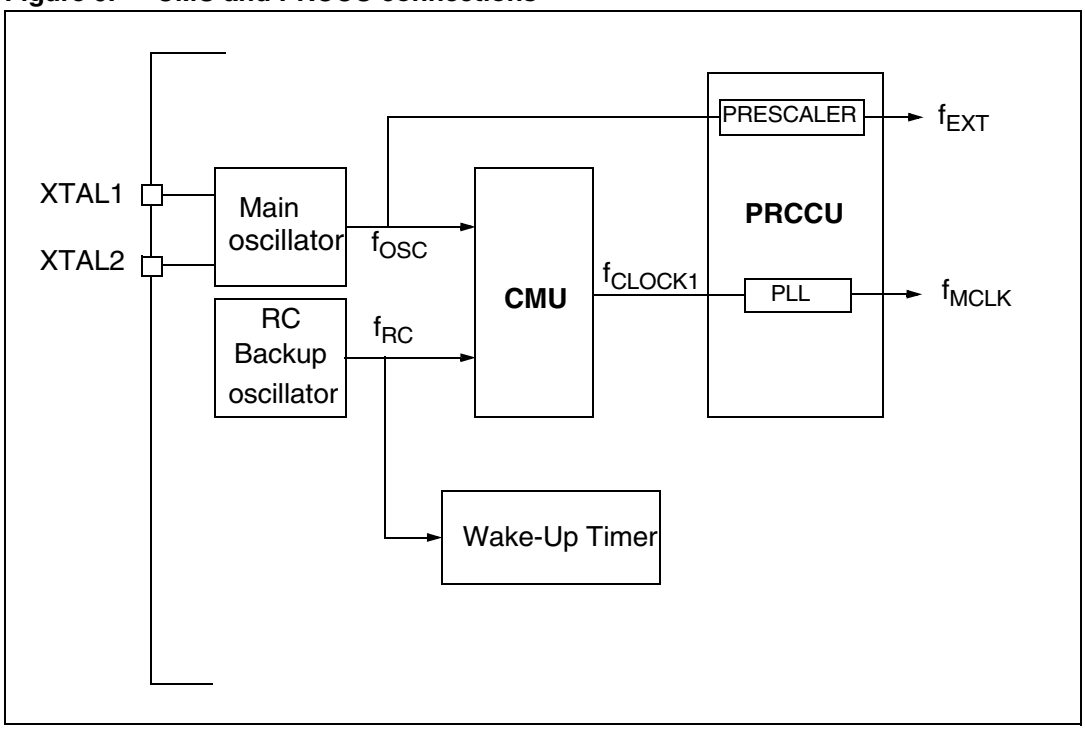
3.3 Clocks

The STR73x internal clocks are derived from two sources:

- External crystal (4 - 8 MHz), connected to the XTAL1 and XTAL2 pins, or an external pulse generator connected to XTAL2 (XTAL1 pin tied to V_{SS})
- On-chip RC Oscillator or Backup oscillator (2 MHz or 32 kHz)

The two clock sources are managed by the CMU (Clock Monitor Unit).

Figure 9. CMU and PRCCU connections



At power-on, the CMU automatically selects the RC Backup Oscillator (f_{RC} running at 2 MHz) as the input clock to the Reset and Clock Control Unit (PRCCU). Driven by the RC oscillator, the STR73x can start executing code while the main oscillator clock (f_{OSC}) is stabilizing.

For the same reason, the CMU also selects f_{RC} when the STR73x wakes-up from Stop and Halt modes. The CMU can generate an interrupt after a programmable delay, allowing software to switch over to the main oscillator when f_{OSC} is stable. By default, the delay is approximately 5 ms at 8 MHz f_{OSC}.

The CMU switches automatically to the backup oscillator when a failure is detected on f_{OSC}.

The internal RC-Oscillator drives the Wake-up Timer directly. All the other parts of the device work with $f_{MCLK,}$ optionally prescaled by frequency dividers specific to each peripheral and gated by the CFG_PCGR0/PCGRB0 and CFG_PCGR1/PCGRB1 registers.

f_{EXT} can be selected by most timers as an input clock. It supplies an alternate time base from the PLL clock frequency (f_{MCLK}). It has the advantage of providing regular time frames unaffected by any scaling effect if f_{MCLK} is slowed-down to reduce power consumption.

3.3.1 RC oscillator

The on-chip RC Oscillator provides two different operating modes:

- Low frequency, low current mode.
  In this mode the Oscillator has to be biased through the V_{BIAS} pin by a 1.3 MΩ external resistor connected to ground. The oscillator works at 32 kHz.
- High frequency, high current mode.
  In this mode the Oscillator is biased through an internal bias branch. The oscillator works at 2 MHz. This is the operating mode after Reset and wake-up from STOP and Halt mode.

RC Oscillator mode can be associated with RUN and STOP modes of the STR73x using the RCFR and RCFS bits in the CMU_CTRL register.

3.4 Low power modes

3.4.1 Slow mode

In Slow mode, you reduce power consumption by slowing down the main clock. You can continue to use all the device functions of the chip, but at reduced speed. The MCLK frequency is changed to f_{CLOCK2} or $f_{CLOCK2}/16$.

To enter and exit Slow mode, toggle the CSU_CKSEL and CK2_16 bits in the PRCCU_CFR register. Selecting Slow mode turns off the PLL automatically. A maskable interrupt is available, which is generated whenever the CK2_16 bit is switched.

3.4.2 WFI mode

In WFI mode, you reduce power consumption by stopping the core. The program stops executing, but peripherals are kept running and the register contents are preserved. The device resumes, and execution restarts when an interrupt request is sent to the EIC.

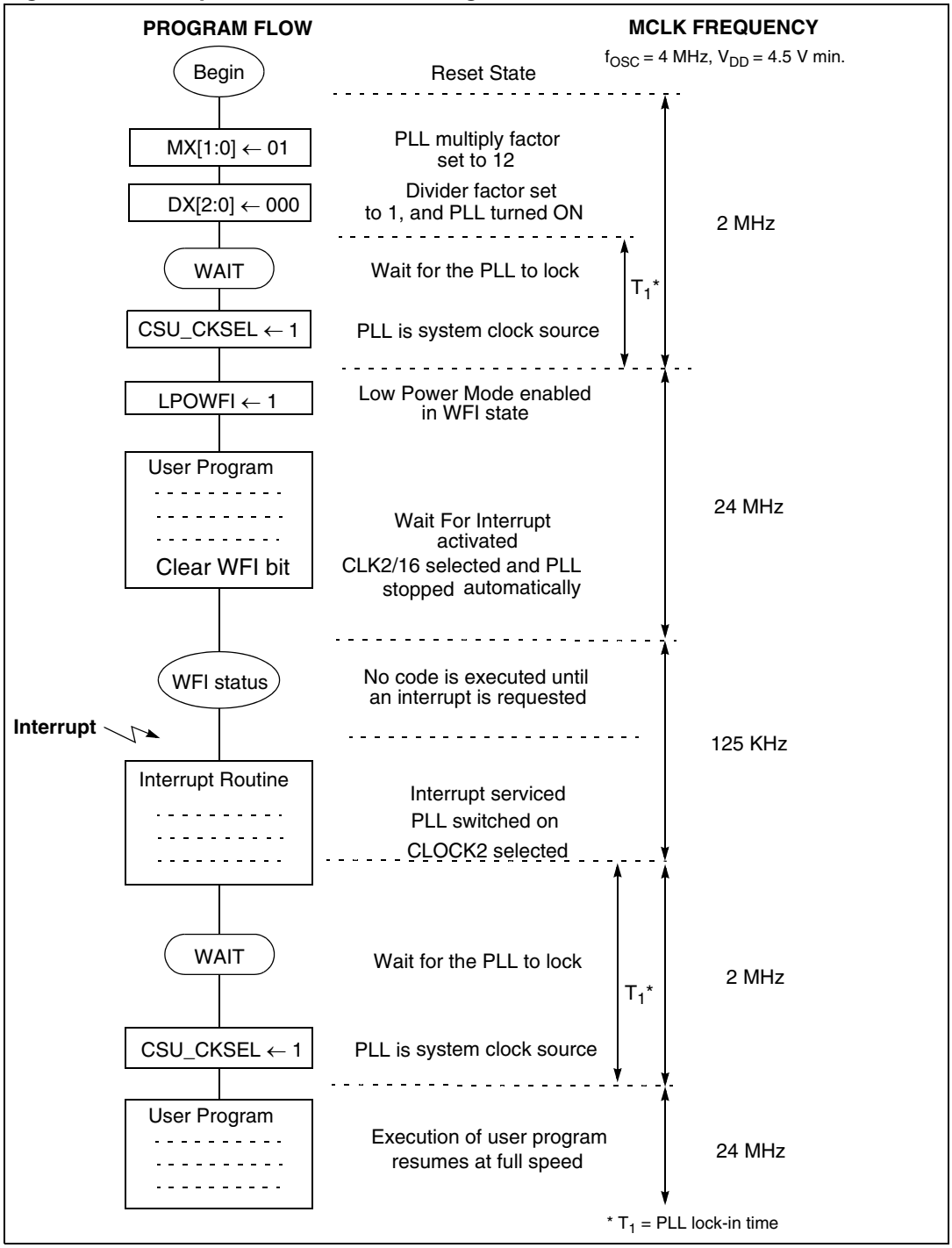
To enter WFI mode, clear the WFI bit in the PRCCU_SMR register.

To wake-up from WFI mode an interrupt request must be acknowledged by the EIC.

3.4.3 LPWFI mode

LPWFI (Low power wait for interrupt) is a combination of WFI and Slow modes. Refer to *Figure 10* for an example.

Figure 10. Example of LPWFI mode using CLK2/16



To enter LPWFI mode, first configure the WFI_CKSEL and LOPWFI bits in the PRCCU_CCR to select the clock speed to be used in LPWFI mode.

Then clear the WFI bit in the PRCCU_SMR register.

To wake-up from LPWFI mode, an interrupt request must be acknowledged by the EIC. MCLK then switches back automatically to CLOCK2.

3.4.4 Halt mode

In Halt mode, all the oscillators present on the device are stopped, the power consumption is almost nil (only leakage current).

To enter Halt mode first set the EN_HALT bit in the PRCCU_SMR register and clear the SRESEN bit in the PRCCU_CCR register. Then set the HALT bit in the PRCCU_CCR register.

Wake-up from Halt mode is only possible by means if an external or LVD reset.

3.4.5 Stop mode

Stop mode stops all the oscillators without resetting the device, preserving the device status (except the CSU_CKSEL and the STOP_I bits in PRCCU_CFR register).

To enter Stop mode, perform the Stop Sequence via the Wake-Up Unit WIU (refer to *Section 7.9*). The device will remain in Stop mode until a wake-up line is asserted to restart program execution.

On wake-up from Stop mode, the CMU automatically selects the 2 MHz RC-Oscillator clock as input clock to the PRCCU.

When the wake-up event is acknowledged, user code execution restarts 400 RC-Oscillator cycles after the wake-up event (equivalent to 200 µs with a 2 MHz RC-Oscillator clock frequency).

On wake-up from Stop mode, the STOP_I bit in the PRCCU_CFG register is set, to indicate that a wake-up from Stop mode has occurred. An interrupt is generated, if enabled

Reset has priority over Stop; so, if the system is in Stop mode and a reset occurs, the oscillator restarts and goes through the reset sequence. A new Stop condition can occur only after MCLK restarts.

3.5 Clock monitor unit (CMU)

3.5.1 Introduction

The CMU has three input clocks:

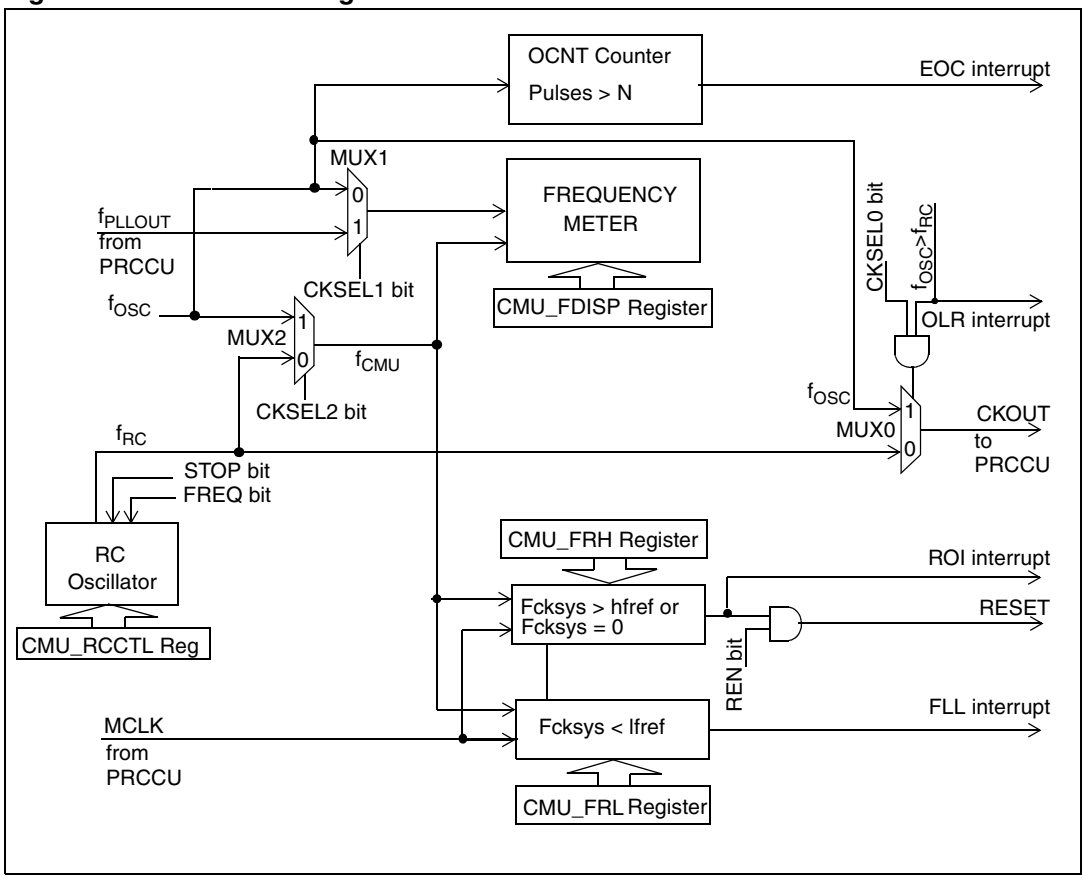
- f_{OSC} from the main oscillator
- f_{RC} from the backup oscillator
- MCLK from the PRCCU

CKOUT is the CMU output clock to the PRCCU.

When the CMU is powered on, f_{RC} clock is sent out as CKOUT by default.

Figure 11 shows the block diagram of the Clock Monitor Unit.

Figure 11. CMU block diagram



3.5.2 Register write protection

All CMU registers are write protected. To modify their register content, you have to enable write access by writing a sequence of two consecutive keywords in the CMU_WE register.

3.5.3 Clock source selection

After external reset has elapsed, OCNT counter counts the f_{OSC} pulses and when it reaches N (where N is a constant defined as 512*EOCV (End Of Count Value is defined in the EOCV register) it sets the EOC bit and verifies the status of System Reset.
If System Reset is still active, the CKOUT is automatically switched to the main clock (f_{OSC}) and the CKSEL0 bit in the CMU_CTRL register is set by hardware.
You can modify the CKSEL0 bit to switch by software from f_{OSC} to f_{RC} and vice versa.

3.5.4 Oscillator frequency monitoring

If f_{OSC} is selected as CKOUT, the CMU automatically checks if it is greater than f_{RC} (usually this is always true) if not, f_{RC} is automatically selected as CKOUT, the CKSEL0 bit in CMU_CTRL register is forced to 0, the OLR bit is set and an interrupt is generated if enabled.

3.5.5 MCLK frequency monitoring

The CMU also checks the System clock frequency MCLK. If it is greater than a reference value defined in the CMU_FRH (Frequency Reference High) register or if it is off, a reset signal is generated (if enabled by setting the REN bit in the CMU_CTRL register). The ROI bit is set and an interrupt (ROI) is generated if enabled.

If it is less than a reference value defined in the CMU_FRL (Frequency Reference Low) register, the FLL bit is set and an interrupt is generated if enabled.

Caution: It is strongly recommended to disable reset generation before switching the CLKSEL2 bit (f_{CMU} clock). To avoid an unwanted System Reset, you should wait for at least 16 f_{CMU} pulses and verify the status of the RON bit in the CMU_STAT register and the value of the CMU_FRH and FRL registers before re-enabling reset generation.

3.5.6 Clock frequency measurement

A simple Frequency Meter allows software to read a rough value of f_{OSC} or f_{PLLOUT} depending on the value of the CKSEL1 bit. To start measurement, set the SFM (Start Frequency Measurement) bit in the CMU_CTRL register. When the measurement is done hardware clears the SFM bit. You can then read the frequency value is ready in the CMU_FDISP (Frequency Display) register.The meaning of the read value is explained in the register description.

Both the Frequency Meter and the System Clock checking logic are driven by default by f_{RC} but the software can swap to f_{OSC} by programming the CKSEL2 bit in the CMU_CTRL register.

The CKON2,CKON1 and CKON0 bits indicate which is the actual clock at the output of the corresponding multiplexer.

3.5.7 RC oscillator control

You can adjust the frequency of the RC oscillator by programming the CMU_RCCTL register.

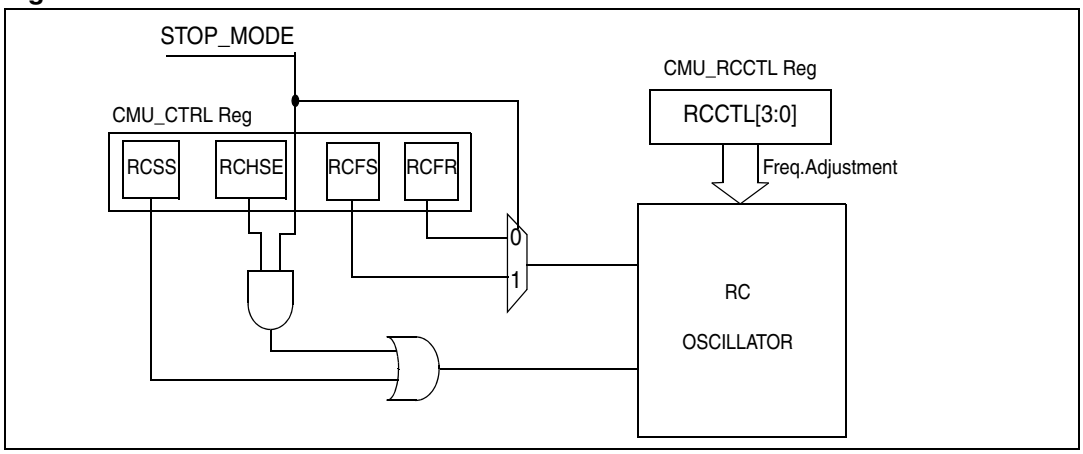
To choose the frequency of the RC oscillator between high an low in both Run mode and Stop mode, program the RCFR and RCFS bits in the CMU_CTRL register. The RCFR bit is reset in Stop mode.

The RC oscillator can be switched off in Stop mode by setting the RCHSE bit in the CMU_CTRL register or it can be switched off in Run mode by setting the RCSS bit in the CMU_CTRL register.

Figure 12 shows an overview of the RC Oscillator control bits.

The CMU is asynchronously reset by the OR of all reset sources which input the system reset control unit and by the STOP signal and it is synchronously reset by the reset generated by itself.

Figure 12. RC oscillator control bits



3.5.8 Limitations

The RC oscillator frequency must be less than the crystal or resonator oscillator frequency ($f_{RC} < f_{OSC}$).

Frequency checking works properly only if $f_{CMU} < f_{MCLK}$ else the checking must be disabled loading the CMU FRH register with FFFh and the CMU_FRL register with 000h.

3.5.9 Register description

Reserved bits cannot be written and are always read as 0.

The registers can not be accessed by byte.

All registers are reset by System Reset and Stop signal except the CMU_RCCTL register and the OSCS, RCFS, and RCHSE bits in the CMU_CTRL register, which are reset only by System Reset or on wake-up from Stop mode.

3.5.10 RC oscillator control register (CMU_RCCTL)

Address Offset: 00h

Reset value: 0008h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												RCCTL[3:0]			
-												rw	rw	rw	rw

Bits 15:4 = Reserved, must be kept at reset value (0).

Bits 3:0 = **RCCTL[3:0]:** *RC oscillator Control bits.*
This value adjusts the frequency of RC oscillator.

3.5.11 Frequency display register (CMU_FDISP)

Address Offset: 04h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				FD[11:0]											
-				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 = Reserved, must be kept at reset value (0).

Bits 11:0 = **FD[11:0]:** *Measured Frequency bits.*
This register displays the measured frequency (f_{IN}) of f_{OSC} or f_{PLL} using f_{CK} as a base. The measured value is given by the following formula: f_{IN} = (FD[11:0]/16) * f_{CK}, where f_{CK} is the clock for the digital logic.

3.5.12 Frequency reference high register (CMU_FRH)

Address Offset: 08h

Reset value: 0FFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				FH[11:0]											
-				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 = Reserved, must be kept at reset value (0).

Bits 11:0 = **FH[11:0]:** *Frequency reference High bits.*
When the frequency of CKSYS is higher than FH[11:0] value, a System Reset or an interrupt can be generated depending on the REN bit in the CMU_CTRL register.
The reference value is given by: (FH[11:0]/16) * FCK, where Fck is the frequency of clock for digital logic.

3.5.13 Frequency reference low register (CMU_FRL)

Address Offset: 0Ch

Reset value 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				FL[11:0]											
-				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 = Reserved, must be kept at reset value (0).

Bits 11:0 = **FL[11:0]:** *Frequency reference Low bits.*
When the frequency of MCLK is lower than FL[11:0] value, an interrupt is generated.
The reference value is given by: (FL[11:0]/16) * Fck, where Fck is the frequency of clock for digital logic.

3.5.14 Control register (CMU_CTRL)

Address Offset: 10h

Reset value 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					OSCS	CRFR	RCFR	RCFS	RCHSE	RCSS	SFM	REN	CKSEL2	CKSEL1	CKSEL0
-					rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:11 = Reserved, must be kept at reset value (0).

Bit 10 = **OSCS:** *Oscillator Stop bit*
This bit is set by software to stop the Main oscillator. It is cleared by hardware on wake-up from Stop mode.
0: No effect
1: Stop Main Oscillator (f_{OSC}) if is not selected as CKOUT.

Bit 9 = **CRFR:** *CMU Reset Flag Reset bit*
This bit is set by software to clear the CRF bit in the CMU_STAT register.
0: No effect
1: Clear CRF flag

Bit 8 = **RCFR:** *RC oscillator Frequency in Run mode*
See *Figure 12*. This bit is set and cleared by software to select the RC oscillator frequency in Run mode.
0: RC oscillator frequency in Run mode is high.
1: RC oscillator frequency in Run mode is low.

Bit 7 = **RCFS:** *RC oscillator Frequency in Stop mode*
See *Figure 12*. This bit is set by software to select the RC oscillator frequency when the device next enters Stop mode. It is cleared by hardware on wake-up from Stop mode.
0: RC oscillator frequency in Stop mode is high.
1: RC oscillator frequency in Stop mode is low.

Bit 6 = **RCHSE:** *RC oscillator Hardware Stop Enable bit*
See *Figure 12*. This bit is set by software to stop the RC oscillator when the device next enters Stop mode. It is cleared by hardware on wake-up from Stop mode.
0: No effect
1: Stop Backup Oscillator when next entering Stop mode.

Bit 5 = **RCSS:** *RC oscillator Software Stop*
See *Figure 12*. This bit is set by software in Run mode to stop the Backup Oscillator. It is cleared by hardware in Stop mode.
0: No effect
1: Stop Backup Oscillator if is not selected as CKOUT.

Bit 4 = **SFM:** *Start Frequency Measurement*
This bit is set by software to start a clock frequency measurement. It is cleared by hardware when the measurement is ready in the CMU_FDISP register.
0: Measurement ready (read only, write has no effect)
1: Start measurement

Bit 3 = **REN:** *CMU Reset Enable*
This bit is set and cleared by software. It can be used to enable reset generation when the RON bit gets set.
0: CMU Reset generation disabled
1: CMU Reset generation enabled

Bit 2 = **CKSEL2:** *CMU clock selection*
This bit is set and cleared by software to select the clock source for driving the CMU logic (f_{CMU}).
0: Select Backup Oscillator (f_{RC})
1: Select Main Oscillator (f_{OSC})

Bit 1 = **CKSEL1:** *Oscillator-PLL selection*
This bit is set and cleared by software to select the clock source for the Frequency Meter.
0: Select Main Oscillator (f_{OSC})
1: Select PLL output from PRCCU (f_{PLLOUT})

Bit 0 = **CKSEL0:** *RC-Oscillator selection*
This bit is set and cleared by software to select the clock source for CKOUT. It is also cleared by hardware when the Main Oscillator is detected to be off and set by hardware if the System Reset is still active when the EOC event occurs.
0: Select Backup Oscillator (f_{RC})
1: Select Main Oscillator (f_{OSC})

Note: The CKON bits in the CMU_STAT Register are updated only a clock switch has become effective and after a few CKOUT pulses.

A clock switch becomes effective only if the selected clock is on.

3.5.15 Status register (CMU_STAT)

Address Offset: 14h

Reset value 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											CRF	RON	CKON 2	CKON 1	CKON 0
-											r	r	r	r	r

Bits 15:5 = Reserved, must be kept at reset value (0).

Bit 4 = **CRF:** *CMU Reset Flag bit*
This bit is set by hardware. It can be cleared by software setting the CRFR bit in the CMU_CTRL register.
0: No reset caused by CMU
1: Reset was caused by CMU

Bit 3 = **RON:** *CMU Reset condition ON status bit*
This bit is set and cleared by hardware.
0: No RON condition
1: RON condition detected. $f_{MCLK} > (FH[11:0]/16) * f_{CMU}$ or MCLK off. A reset is generated if the REN bit is set. An interrupt is generated if the ROIM bit is set.

Bit 2 = **CKON2:** *MUX2 status bit*
This bit is set and cleared by hardware. It indicates which clock drives f_{CMU}.
0: f_{RC}
1: f_{OSC}

Bit 1 = **CKON1:** *MUX1 status bit.*
This bit is set and cleared by hardware. It indicates which clock drives the Frequency Meter
0: f_{PLLOUT}
1: f_{OSC}

Bit 0 = **CKON0:** *MUX0 status bit.*
This bit is set and cleared by hardware. It indicates which clock drives CKOUT.
0: f_{RC}
1: f_{OSC}

3.5.16 Interrupt status register (CMU_IS)

Address Offset: 18h

Reset value 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												ROI	FLL	EOC	OLR
-												rw	rw	rw	rw

Bits 15:4 = Reserved, must be kept at reset value (0).

Bit 3 = **ROI:** *Reset ON Interrupt pending bit*
This bit is set by hardware and cleared by software
0: No ROI interrupt pending
1: ROI interrupt pending. It is set when the RON bit is set.

Bit 2 = **FLL:** *Clock Frequency Less than Low reference pending bit.*
This bit is set by hardware and cleared by software
0: No FLL interrupt pending
1: FLL interrupt pending. f_{MCLK} less than LFREF value.

Bit 1 = **EOC:** *End of Counter pending bit.*
This bit is set by hardware and cleared by software
0: No EOC interrupt pending
1: EOC interrupt pending. The number of f_{OSC} pulses has reached EOCV[7:0]*512.

Bit 0 = **OLR:** *Oscillator frequency Less than RC frequency pending bit*
This bit is set by hardware and cleared by software
0: No OLR interrupt pending
1: OLR interrupt pending. The Main oscillator frequency (f_{OSC}) is less than the frequency of the Backup oscillator (f_{RC}).

3.5.17 Interrupt mask register (CMU_IM)

Address Offset: 1Ch

Reset value 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												ROIM	FLLM	EOCM	OLRM
-												rw	rw	rw	rw

Bits 15:4 = Reserved, must be kept at reset value (0).

Bit 3 = **ROIM:** *Reset ON Interrupt Mask bit*
This bit is set and cleared by software.
0: ROI interrupt disabled
1: ROI interrupt enabled

Bit 2 = **FLLM:** *Clock Frequency Less than Low reference interrupt Mask bit*
This bit is set and cleared by software.
0: FLL interrupt disabled
1: FLL interrupt enabled

Bit 1 = **EOCM:** *End of Counter interrupt Mask bit*
This bit is set and cleared by software.
0: EOC interrupt disabled
1: EOC interrupt enabled

Bit 0 = **OLRM:** *Oscillator frequency Less than RC frequency interrupt Mask bit*
This bit is set and cleared by software.
0: OLR interrupt disabled
1: OLR interrupt enabled

3.5.18 End of count value register (CMU_EOCV)

Address Offset: 20h

Reset value -> M

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								EOCV[7:0]							
-								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 = Reserved, must be kept at reset value (0).

Bits 7:0 = **EOCV[7:0]:** *End Of Count Value*
This byte is written by software. When the Oscillator Counter (OCNT) reaches the value EOCV[7:0]*512, an EOC interrupt is generated if EOCM is set.

3.5.19 Write enable register (CMU_WE)

Address Offset: 24h

Reset value 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WEK[15:0]															
w															

Bits 15:0 = **WEK:** *Write Enable Key value*
To enable write access to all CMU registers:

- Write the hex. key value “50FA” and then the hex. key value “AF05”.

To disable write access:

- Write any other value in this register.

3.5.20 CMU register map

Table 5. CMU peripheral register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	CMU_RCCTL	Reserved												RCCTL[3:0]			
04h	CMU_FDISP	Reserved				FD[11:0]											
08h	CMU_FRH	Reserved				FH[11:0]											
0Ch	CMU_FRL	Reserved				FL[11:0]											
10h	CMU_CTRL	Reserved					OSCS	CRF	RCFR	RCFS	RCHSE	RCSS	SFM	REN	CKSEL2	CKSEL1	CKSEL0
14h	CMU_STAT	Reserved											CRFR	RON	CKON2	CKON1	CKON0
18h	CMU_IS	Reserved												ROI	FLL	EOC	OLR
1Ch	CMU_IM	Reserved												ROIM	FLLM	EOCM	OLRM
20h	CMU_EOCV	Reserved								EOCV[7:0]							
24h	CMU_WE	WEK[15:0]															

See *Table 2* for the base address

3.6 Power, reset and clock control unit (PRCCU)

The PRCCU generates the internal clocks for the CPU and for the on-chip peripherals. The Clock Control Unit input clock is generated by the CMU module please refer to *Section 3.5 on page 32*)

3.6.1 Overview

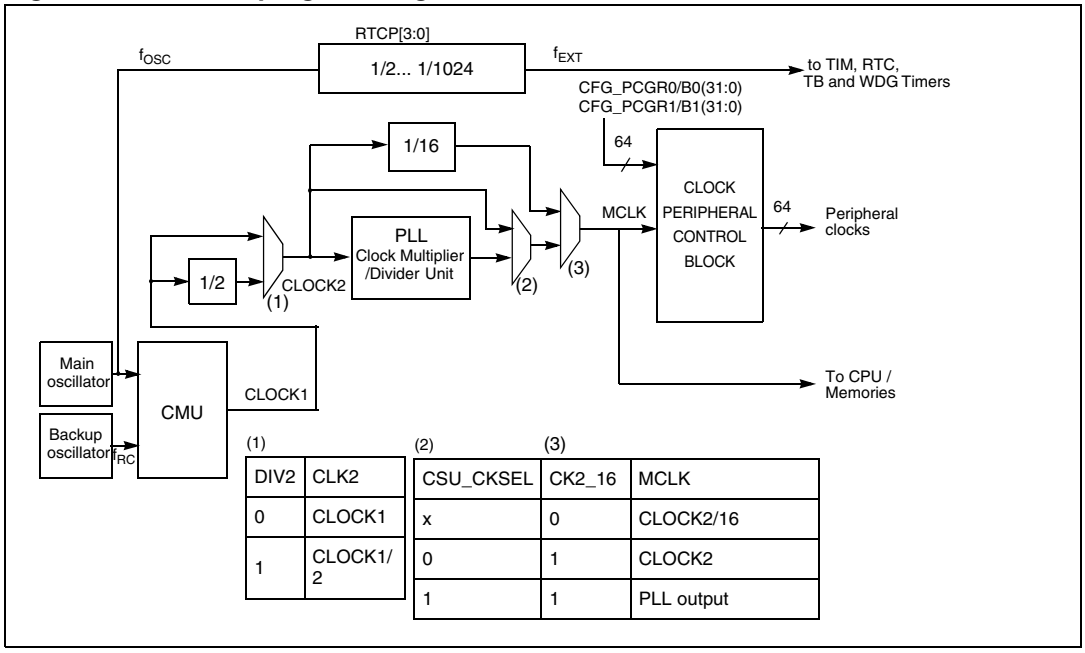
As shown in *Figure 13*, you can divide the CLOCK1 input clock signal by two by selecting a 1/2 divider. The resulting signal, CLOCK2, is the reference input clock to the programmable PLL which is capable of multiplying the clock frequency by a factor of 12, 16, 20 or 28. The multiplied clock is then divided by a programmable divider, by a factor of 1 to 7. The range of available multiplication and division factors allows a wide range of operating clock frequencies to be derived from a single crystal or resonator frequency.

The PLL can be turned off. In this case, the MCLK signal may be programmed as CLOCK2 divided by 16.

The internal system clock, MCLK, is routed to all on-chip peripherals. Under software control, the peripheral clock to each peripheral can be gated through the PCGR0/B0 and PCGR1/B1 Registers (see *Section 4 on page 53*).

Each time a clock change occurs, it takes several cycles to complete.

Figure 13. PRCCU programming



(1)

DIV2	CLK2
0	CLOCK1
1	CLOCK1/ 2

(2) (3)

CSU_CKSEL	CK2_16	MCLK
x	0	CLOCK2/16
0	1	CLOCK2
1	1	PLL output

3.6.2 PLL clock multiplier programming

The CLOCK1 signal generated by the CMU drives a programmable divide-by-two circuit. If the DIV2 control bit in the PRCCU_CFR register is set (reset condition), CLOCK2, is equal to CLOCK1 divided by two; if DIV2 is reset, CLOCK2 is identical to CLOCK1. In practice, the divide-by-two is used in order to ensure a 50% duty cycle signal.

When the PLL is active, it multiplies CLOCK2 by 12, 16, 20 or 28, depending on the status of the MX[1:0] bits in the PRCCU_PLLCR register. The multiplied clock is then divided by a factor in the range 1 to 7, determined by the status of the DX[2:0] bits. When these bits are programmed to 111, the PLL is switched off (default condition after reset).

The frequency multiplier contains a frequency comparator between CLOCK2 and the PLL clock output that verifies if the PLL clock has stabilized (locked status). The LOCK bit in PRCCU_CFR register becomes 1 when this condition occurs and maintains this value as long as the PLL is locked, going back to 0 if for some reasons (change of MX bits value, stop and restart of PLL or CLOCK2 and so on) the PLL loses the programmed frequency in which it was locked. It is possible to select the PLL clock as system clock only when the LOCK bit is '1'. If the LOCK bit returns to '0' the system clock switches back to CLOCK2 even if the CSU_CKSEL bit is '1'. The PLL selection is further conditioned by the status of the main voltage regulator: the PLL can be selected only when VROK bit in VR_CTR register is '1', that is when the Voltage Regulator is providing a stable supply voltage). Setting the CSU_CKSEL bit in the PRCCU_CFR register allows to select the multiplier clock as system clock, but the two conditions mentioned above must be matched (PLL locked and Voltage Supply stable).

The PLL is able to provide a low precision free running frequency from 125kHz to 625kHz, usable to slow down the program execution. The FREEN and DX[2:0] bits in the PRCCU_PLLCR register enable this mode: when PLL is off and FREEN bit is '1', that is, all these four bits are set, the PLL provides this clock. The selection of this clock is still managed by the CSU_CKSEL bit, but is not conditioned by the LOCK bit in the PRCCU_CFR register and the VROK bit in the VR_CTR register. To avoid unpredictable behavior of the PLL clock, the user must set and reset the Free Running mode only when the PLL clock is not the system clock, i.e when the CSU_CKSEL bit is '0'.

Care is required, when programming the PLL multiplier and divider factors, not to exceed the maximum allowed operating frequency for MCLK.

When selected, the PLL can go into Free Running mode even when the reference clock signal disappears, and continue to provide a valid clock signal to the system (even though not precise and with a possible wide spread, due to temperature, process and supply voltage variations). The free running frequency is not predetermined or fixed, but depends on the current PLL setting, since the Voltage Controlled Oscillator (VCO) determines the signal frequency (in the range 1-5MHz), that is prescaled by the following set of digital dividers.

Example 1:

- f_{OSC} = 4 MHz
- MCLK = 20 MHz (MX[1:0] = '00' - DX[2:0] = '001' - DIV2 = '1' - FREF_RANGE = '0')

  With this configuration, the expected free running frequency in case the 4MHz reference disappears, could be in the range of 0.5-2.5 MHz (divider by 2 enabled in DX[2:0]).

Example 2:

- f_{OSC} = 4 MHz
- MCLK = 14 MHz (MX[1:0] = '10' - DX[2:0] = '011' - DIV2 = '1' - FREF_RANGE = '0')

  With this configuration, the expected free running frequency in case the 4 MHz f_{OSC} disappears, could be in the range of 0.25-1.25 MHz (divider by 4 enabled in DX[2:0]).

3.6.3 Peripheral clocks

The system clock, MCLK, which may be the output of the PLL clock multiplier, CLOCK2, or CLOCK2/16, is also routed to all on-chip peripherals and acts as the central time base for all timing functions. At the exit of reset sequence only the CPU, the memory and a small subset of the peripherals present on the chip start working. The remaining part of the system (depending on the device) is however stopped because the related PCGR bits are reset. To start them, the user has to write '1' into the related register bit of Peripheral Clock Gating Registers (CFG_PCGR0/B0, CFG_PCGR1/B1).

The clock provided to such peripheral or group of peripherals can be stretched again, writing at '0' the related bit of the CFG_PCGR0/B0, CFG_PCGR1/B1. This allows to have at one time only the desired peripherals operating and to start an additional one only when necessary. Depending on the desired configuration (please refer to *Section 4 on page 53*), a peripheral can be kept under reset, as long as the clock is stretched. It is up to the user to reconfigure this one after the recovering of the related clock.

Note: After enabling a peripheral previously forced in reset mode (by setting the corresponding bit in the CFG_PCGR0/B0, CFG_PCGR1/B1 registers), the clock of the peripheral is running but, during the first 8 clock cycles, the peripheral itself is kept under reset. The user must wait at least this period of time before starting to program the peripheral.

3.6.4 RT clock (f_{EXT})

A programmable divider is available to provide a real-time clock frequency based on the main oscillator. The f_{OSC} signal is divided by a factor in the range 2-2^10, depending on the PRCCU_RTCPR register value.

3.6.5 Clock configuration reset state

In power-on reset state, the PRCCU_CFR value is 8008h. Consequently, in reset state the clock configuration is DIV2 = 1, CK2_16=1 and therefore MCLK operates at the external main clock frequency (f_{OSC}) divided by 2.

3.6.6 Interrupt generation

The PRCCU generates an interrupt request on the following events:

Table 6. PRCCU interrupts

Event	Description	Event trigger	Interrupt mask	Event flag
CLK2/16 Switching	CLK2/16 selected or deselected as RCLK source	CK2_16 bit in PRCCU_CFR register toggles	EN_CK2_16 bit in PRCCU_CCR register	CK2_16_I bit in PRCCU_CFR register
Lock	PLL becomes locked or unlocked	LOCK bit in PRCCU_CFR register toggles	EN_LOCK bit in PRCCU_CCR register	LOCK_I bit in PRCCU_CFR register
Stop	CLK restarts after waking up from Stop mode		EN_STOP bit in PRCCU_CCR register	STOP_I bit in PRCCU_CFR register

When any of these events occur, the corresponding pending bit in the PRCCU_CFR register becomes '1' and the interrupt request is forwarded to the interrupt controller. It is up to the

user to reset the pending bit as the first instruction of the interrupt routine. The pending bits are clear-only (cleared only by writing '1'). Each interrupt can be masked by resetting the corresponding mask bit in the PRCCU_CCR register.

Table 7. Operating modes using main crystal controlled oscillator

MODE	MCLK	DIV2	CSU_CKSEL	MX[1:0]	DX[2:0]	CK2_DIV16
PLL x 28	CLOCK1 / 2 x (14 / D)	1	1	1 0	D-1	1
PLL x 20	CLOCK1 / 2 x (10 / D)	1	1	0 0	D-1	1
PLL x 16	CLOCK1 / 2 x (8 / D)	1	1	1 1	D-1	1
PLL x 12	CLOCK1 / 2 x (6 / D)	1	1	0 1	D-1	1
SLOW 1	CLOCK1 / 2	1	0	X	XXX	1
SLOW 2	CLOCK1 / 32	1	X	X	X	0
WFI	If LPOWFI=0, no changes occur on MCLK, but CPU is stopped.					
LOW-POWER WFI	CLOCK2/16	1	X	X	X	X
RESET	CLOCK1 / 2	1	0	00	111	1

3.6.7 Register description

3.6.8 Clock control register (PRCCU_ CCR)

Address Offset: 00h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				EN_ HALT	EN_ STOP	EN_ CK2_1 6	res.	EN_ LOCK	reserved			SRES EN	res.	WFI_ CKSEL	LPO WFI
				rw	rw	rw	rw	rw				rw	rw	rw	rw

Bits 31:12 = Reserved, always return '0' when read.

Bit 11 = **EN_HALT**: *Halt enable*.
0: Setting HALT bit in PRCCU_SMR register will have no effect
1: Setting HALT bit in PRCCU_SMR will enter Halt mode or generate a Software reset

Bit 10 = **EN_STOP**: *STOP Interrupt Mask*.
0: STOP interrupts disabled
1: STOP interrupts enabled

Bit 9 = **EN_CK2_16**: *CK2_16 Interrupt Mask*.
0: CK2_16 interrupts disabled
1: CK2_16 interrupts enabled

Bit 8 = Reserved, must be kept at reset value (0).

Bit 7 = **EN_LOCK**: *LOCK Interrupt Mask*.
0: LOCK interrupts disabled
1: LOCK interrupts enabled

Bits 6:4 = Reserved, always return ‘0’ when read.

Bit 3 = **SRESEN**: *Software Reset Enable.*
0: Halt mode is entered when the HALT bit is set
1: A Reset is generated when the HALT bit is set.

Bit 2 = Reserved, must be kept at reset value (0).

Bit 1= **WFI_CKSEL**: *WFI Clock Select*.
This bit selects the clock used in LPWFI mode if LPOWFI = 1.
0: MCLK during LPWFI mode is CLOCK2/16
1: Reserved

Bit 0 = **LPOWFI**: *Low Power Wait For Interrupt mode*.
0: LPWFI mode disabled. When WFI is executed, the CPU is stopped and MCLK is unchanged
1: LPWFI mode enabled. The device enters LPWFI mode when the WFI instruction is executed.

3.6.9 Voltage regulator control register (PRCCU_VRCTR)

Address Offset: 04h
Reset value: 0000 0014h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved											VR LPW	VR OFF_ REG	VROK	reserved	
-											rw	rw	r	rw	

Bits 31:5 = Reserved, always return ‘0’ when read.

Bit 4 = **VRLPW**: *Voltage Regulator for Low Power WFI.*
This bit is set and cleared by software.
0: Main VR is switched off when the system enters low power mode.
1: Main VR stays on when the system enters low power mode. Low power mode is defined as any of the following states:

- Low power WFI
- CK2_16 bit = 0 (CLOCK2/16 is system clock set by the user).

Note: If the Main Voltage Regulator is switched-off during LPWFI it is recommended to:
1. Disable DMA transfers to/from RAM memory (to avoid voltage drop during LPWFI mode).
2. Enter LPWFI mode fetching from FLASH memory (to avoid voltage drop when the system resumes from LPWFI).

Bit 3 = **VROFF_REG**: Voltage Regulator OFF state.
This bit is set and cleared by software.
0: Main VR on

1: Main VR off. In this state the Main Regulator has zero power consumption, and the PLL is automatically deselected.

Note: The Main VR is also switched off in the following conditions:

Stop mode

Halt mode

If the VRLPW bit = 0 and the system is in one of the following states:

LPWFI mode

CK2_16 bit = 0 in the PRCCU_CFR register (CLOCK2/16 is system clock set by user).

Bit 2 = **VROK**: *Voltage Regulator OK.*
This bit is set and cleared by hardware.
0: VR not ready
1: VR stabilized

Bits 1:0 = Reserved, must be kept at reset value (“00”).

3.6.10 Clock flag register (PRCCU_CFR)

Address Offset: 08h
Reset Value: 0000 8048 after a Watchdog Reset
Reset Value: 0000 8028 after a Software Reset
Reset Value: 0000 8108 after an Internal LVD Reset (Flash)
Reset Value: 0000 8008 after a Power-On Reset and HW Reset

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV2	STOP_I	CK2_16_I	res.	LOCK_I	reserved		LVD_INT	reserved	WDGRES0	SOFTRES	res.	CK2_16	res.	LOCK	CSU_CKSEL
rw	rc	rc	-	rc	-		r	-	r	r	rw	rw	r	r	rw

Bits 31:16 = Reserved, must be kept at reset value (0).

Bit 15 = **DIV2**: *CLOCK1 Divided by 2.*
This bit controls the divide-by-2 circuit which operates on the CLOCK1 signal.
0: No division of CLOCK1 frequency.
1: CLOCK1 is divided by 2.

Bit 14 = **STOP_I**: *STOP Interrupt pending bit.*
This bit is clear only.
0: No STOP interrupt request pending.
1: STOP Interrupt request pending.

Bit 13 = **CK2_16_I**: *CK2_16 switching Interrupt pending bit.*
This bit is clear only.
0: No CK2_16 Interrupt request pending.
1: CK2_16 Interrupt request pending.

Bit 12 = Reserved, must be kept at reset value (0).

Bit 11 = **LOCK_I**: *LOCK Interrupt pending bit.*
This bit is clear only.
0: No LOCK Interrupt request pending.
1: LOCK Interrupt request pending.

Bits 10:9 = Reserved, must be kept at reset value (0).

Bit 8 = **LVD_INT**: *Internal LVD reset flag.*
This bit is read only.
0: No Internal LVD reset occurred.
1: Internal LVD reset occurred.

Bit 7 = Reserved, must be kept at reset value (0).

Bit 6 = **WDGRES**: *Watchdog reset flag.*
This bit is read only.
0: No Watchdog reset occurred.
1: Watchdog reset occurred.

Bit 5 = **SOFTRES**: *Software Reset Flag.*
This bit is read only.
0: No software reset occurred.
1: Software reset occurred.

Bit 4 = Reserved, must be kept at reset value (0).

Bit 3 = **CK2_16**: *CLOCK2/16 Selection.*
0: CLOCK2/16 is selected and the PLL is off.
1: The input is CLOCK2 (or the PLL output depending on the value of CSU_CKSEL).
This bit is reset by hardware when the system enters LPWFI mode.
An interrupt is generated when this clock is selected and when the application switches from this clock to another available one.

Bit 2 = Reserved, must be kept at reset value (0).

Bit 1= **LOCK**: *PLL locked-in*
This bit is read only.
0: The PLL is turned off or not locked and cannot be selected as system clock source.
1: The PLL is locked.

Bit 0 = **CSU_CKSEL**: *CSU Clock Select*
This bit is set and cleared by software. It is kept reset by hardware when:
- the PLL is off (bits DX[2:0] (PLLCONF) are set to 111);
- the CK2_16 bit (CLK_FLAG) is forced to ‘0’.
0: CLOCK2 provides the system clock
1: The PLL Multiplier provides the system clock if LOCK and VROK bits are ‘1’
If the FREEN bit is set, this bit selects this clock independently by LOCK and VROK bits.

3.6.11 PLL configuration register (PRCCU_PLLCR)

Address Offset: 18h
Reset value: 0000 0007h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	FREEN	FREF_RANGE	MX1	MX0	-	DX2	DX1	DX0
								rw	rw	rw	rw		rw	rw	rw

Bits 31:8 = Reserved, always return ‘0’ when read.

Bit 7 = **FREEN**: *PLL free running mode enable.*
0: Free Running mode disabled. In this case, the PLL operation depends only on the MX[1:0] and DX[2:0] bits.
1: Free Running mode enabled. In this mode, when all three DX[2:0] bits are set, the PLL is not stopped but provides a slow frequency back-up clock, selected by the CSU_CKSEL bit; Operation in this mode not require the LOCK and VROK bits to be set.

Bit 6 = **FREF_RANGE**: *Reference Frequency Range selector bit*
0: Configure PLL for input frequency (CLOCK2) of 1.5-3 MHz
1: Configure PLL for input frequency (CLOCK2) of 3-5 MHz

Note: When an oscillator with a frequency greater than 5 MHz is used, the clock input to the PLL has to be divided by 2 (bit DIV2 set in PRCCU_CFR register) in order to comply with PLL specification.

Bits 5:4 = **MX[1:0]**: *PLL Multiplication Factor.*
Refer to *Table 8* for the MX bit settings.

Table 8. PLL multiplication factors

MX1	MX0	CLK2 x
1	0	28
0	0	20
1	1	16
0	1	12

Bit 3 = Reserved, always return ‘0’ when read.

Bits 2:0 = **DX[2:0]**: *PLL output clock division factor.* Refer to *Table 10* for the DX bit settings.

Table 9. PLL division factors

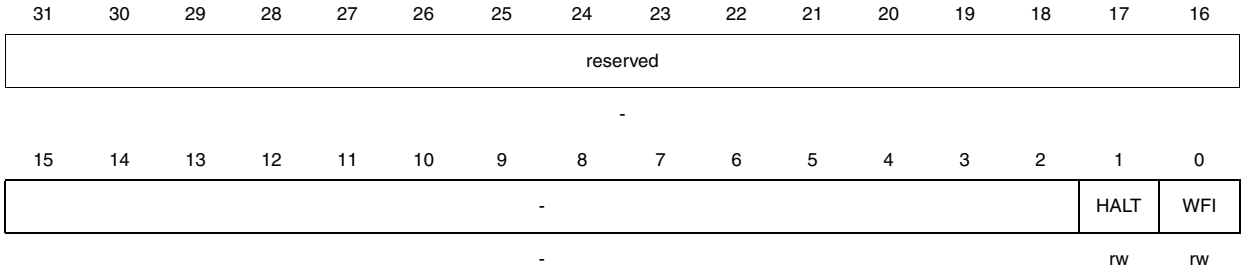
DX2	DX1	DX0	MCLK
0	0	0	PLLCK / 1
0	0	1	PLLCK / 2
0	1	0	PLLCK / 3

Table 9. PLL division factors (continued)

DX2	DX1	DX0	MCLK
0	1	1	PLLCK / 4
1	0	0	PLLCK / 5
1	0	1	PLLCK / 6
1	1	0	PLLCK / 7
1	1	1	CLOCK2 (PLL OFF, Reset State)

3.6.12 System mode register (PRCCU_SMR)

Address Offset: 20h
Reset value: 0000 0001h



Bits 31:2 = Reserved, always return ‘0’ when read.

Bit 1 = **HALT**: *Halt.*
0: No effect
1: Enter Halt mode, or generate a Software reset if the SRESEN bit in the PRCCU_CCR register is set.

Bit 0 = **WFI**: *Wait For Interrupt mode*.
0: Enter WFI (Wait For Interrupt) mode. In this mode the CPU remains in idle state until an interrupt request is acknowledged by the EIC. When this occurs, the bit is set to ‘1’ again. This means that this bit, once reset, can only be set to ‘1’ by hardware.
1: No effect

Caution: If all EIC interrupt channels are masked, clearing this bit will stop program execution indefinitely unless the device is reset. Hence you must ensure that at least one interrupt channel is enabled before clearing the WFI bit.

3.6.13 Real time clock programming register (PRCCU_RTCPR)

Address Offset: 28h
Reset value: 0000 000Fh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												RTCP(3:0)			
-												rw			

Bit 31:4 = Reserved, always return ‘0’ when read.

Bits 3:0 = **RTCP[3:0]**: *Real Time Clock Programming.*
Provides the division factor for the f_{EXT}, according to *Table 10*. When the RTCP[3:0] bits are changed, the prescaler is restarted with the new prescaling factor.

Table 10. Real time clock prescaling factors

RTCP[3:0]	RT clock frequency (f_{EXT})
0000	$f_{OSC}/2$
0001	$f_{OSC}/4$
0010	$f_{OSC}/8$
0011	$f_{OSC}/16$
0100	$f_{OSC}/32$
0101	$f_{OSC}/64$
0110	$f_{OSC}/128$
0111	$f_{OSC}/256$
1000	$f_{OSC}/512$
1001	$f_{OSC}/1024$
1010-1111	stopped

3.6.14 PRCCU register map

Table 11. PRCCU register map

Addr. Offset	Register Name	31:16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	PRCCU_CCR	reserved					EN_HALT	EN_STP	EN_CK2_16	res	EN_LOCK	reserved			SRESEN	res.	WFI_CKSEL	LPOWFI
04h	PRCCU_VRCTR	reserved												VRLPW LPW	VROFF_REG OFF_	VROK	res PR1	res PR0
08h	PRCCU_CFR	reserved	DIV2	STOP_I	CK2_16_I	res	LOCK_I	res.		LVD_INT	res	WDGRES	SOFTRES	res	XT_DIV16	res	LOCK	CSU_CKSEL
18h	PRCCU_PLLCR	reserved									FREEN	FREF_RANGE	MX1	MX0	res	DX2	DX1	DX0
20h	PRCCU_SMR	reserved															HALT	WFI
28h	PRCCU_RTCPR	reserved													RTCP[3:0]			

See *Table 1 on page 17* for the base address

4 Configuration registers (CFG)

4.1 System configuration registers

4.1.1 Configuration register 0 (CFG_R0)

Address Offset: 00h

Reset value: 0000 0000 0000 0000 0000 0xxx x000 0000b

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved					SYS	USER1	USER2	JTBT	res.	DMA BSPI1	DMA BSPI0	reserved			REMAP
-					r	r	r	r	r	rw	rw	rw			rw

Bits 31:11 = Reserved, always return ‘0’ when read.

Bit 10 = **SYS:** *SystemMemory Boot Mode Flag*
This bit is set by hardware after reset.
0: Not in SystemMemory Boot mode
1: Device booted in SystemMemory Boot mode (*Boot memory on page 21*)

Bit 9 = **USER1**: *User1 Boot Mode flag*
This bit is set by hardware after reset.
0: Not in User1 Boot mode
1: Device booted in User1 Boot mode (*Boot memory on page 21*)

Bit 8 = **USER2**: *User2 Boot Mode flag*
This bit is set by hardware after reset.
0: Not in User2 Boot mode
1: Device booted in User2 Boot mode (*Boot memory on page 21*)

Bit 7= **JTBT** *JTAG Boot Mode Flag*
This bit is set by hardware if SystemMemory boot mode is selected and the loader has been stored in RAM via JTAG.
0: Not in SystemMemory Boot mode or Loader not stored in RAM via JTAG
1: Device booted in SystemMemory Boot mode and loader stored in RAM via JTAG

Bit 6 = Reserved, always returns ‘0’ when read.

Bit 5 = **DMABSPI1**: *DMA/BSPI1 Select*.
This bit is set and cleared by software.
0: DMA is selected to transfer data to/from TIM8/TIM9
1: DMA is selected to transfer data to/from BSPI1 (see *Table 24 on page 112*)

Bit 4 = **DMABSPI0**: *DMA/BSPI0 Select*.
This bit is set and cleared by software.
0: Not used
1: DMA is selected to transfer data to/from BSPI0 (see *Table 24 on page 112*).

Bits 3:1 = Reserved, must be kept at reset value (0).

Bit 0 = **REMAP**: *Memory Remapping*.
This bit is set and cleared by software.
0: Internal RAM accessible only at address 0xA000 0000h.
1: Internal RAM remapped to address 0h. It replaces the FLASH memory normally accessible in this location. Even when remapped, the RAM is also accessible at its physical address 0xA000 0000h.

4.1.2 Configuration register 1 (CFG_R1)

Address Offset: 30h

Reset value: 0000 0040h

Bits 31:8 = Reserved, always return ‘0’ when read.

Bits 7:4 = **FLPOD**: *FLASH Power-On Delay*
These bits allow to define a delay before FLASH Power-On, after the system exits from Low Power WFI. This delay is needed if the Main Voltage Regulator is switched off in LPWFI mode (depending on the VRLPW bit in the PRCCU_VRCTR register). The delay should be long enough to ensure that the Main Voltage Regulator is stable before the FLASH comes out of power-down mode. The delay is defined as follows:

On exit from Low Power WFI mode, the RC Oscillator may be running at 2 MHz (TCK2 = 500ns) or at 32 kHz (TCK32 = 31.25us), depending on CMU settings. In the two cases, the delay is equal either to FLPOD[3:0] * 64 * TCK2 or to FLPOD[3:0] * TCK32 (due to internal clock synchronization, up to one TCK2/TCK32 clock cycle is added to the above delay).

Bit 3 = **WUP0S**: *Wake Up Input Line 0 Source Select*
This bit is set and cleared by software to select the wake-up trigger event on Wake-up Input Line 0.
0: WUP0 external pin is connected to Wake-up Unit Input Line 0
1: Internal Wake-up Timer End of Count event is connected to Wake-up Unit Input Line 0

Bit 2 = **EIFILT**: *External Interrupt Filter on Channels INT(15:0)*
This bit is set and cleared by software to enable a digital filter on external interrupt channels INT[15:0]. If enabled, external interrupt channels pulses less than or equal than one system clock cycle are filtered and will not trigger interrupt requests to EIC module.
0: Filter disabled.
1: Filter enabled.

Bits 1:0 = **LPVRCC[1:0]:** *Low Power Voltage Regulator Current Capability*
These bits allow to select the Low-Power Voltage Regulator Current Capability according to the table below.

Table 12. Low power voltage regulator output current

LPVRCC[1:0]	LPVR output current (mA)
00	6
01	4
10	4
11	2

4.1.3 Device identification register (CFG_DIDR)

Address Offset: 34h

Reset value STR73x (144pin) device: EE73 80B5h
Reset value STR73x (100pin) device: EA73 80B4h

This register (read only) returns the device configuration status.

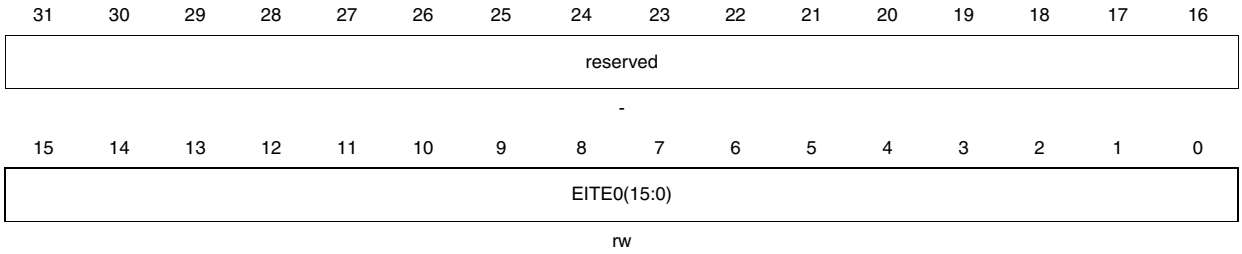
4.2 External interrupt request configuration registers

For an introduction to External interrupts, refer to *Section 7.8: External interrupt pins INT[15:0] on page 102*

4.2.1 External interrupt trigger event register 0 (CFG_EITE0)

Address Offset: 04h

Reset value: 0000 FFFFh



This register determines (together with CFG_EITE1 and CFG_EITE2 registers) which type of event on INTn (n=0,15) input pin will trigger an interrupt request to the EIC module, according to *Table 22 on page 102*.

Bits 31:16 = Reserved, always return ‘0’ when read.

Bits 15:0 = **EITE0[15:0]**: *External Interrupt Channels INT[15:0] Trigger Event Register 0*. Please refer to *Table 22 on page 102* to configure the external interrupt channel INTn (n=0,15).

4.2.2 External interrupt trigger event register 1 (CFG_EITE1)

Address Offset: 24h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EITE1(15:0)															
rw															

This register determines (together with CFG_EITE0 and CFG_EITE2 registers) which type of event on INTn (n=0,15) input pin will trigger the interrupt request to EIC module, according to *Table 22 on page 102*.

Bits 31:16 = Reserved, always return ‘0’ when read.

Bits 15:0 = **EITE1(15:0)**: *External Interrupt Channels INT[15:0] Trigger Event Register 1*. Please refer to *Table 22 on page 102* to configure the external interrupt channel INTn (n=0,15).

4.2.3 External interrupt trigger event register (CFG_EITE2)

Address Offset: 28h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EITE2(15:0)															
rw															

This register determines (together with the CFG_EITE0 and CFG_EITE1 registers) which type of event on INTn (n=0,15) input pin will trigger the interrupt request to the EIC module, according to *Table 22 on page 102*.

Bits 31:16 = Reserved, always return ‘0’ when read.

Bits 15:0 = **EITE2[15:0]**: *External Interrupt Channels INT[15:0] Trigger Event Register 2*. Please refer to *Table 22 on page 102* to configure the external interrupt channel INTn (n=0,15).

4.3 Peripheral clock management registers

4.3.1 Clock management in user mode

Four registers that allow you to switch the clock on or off for each peripheral individually, to reduce overall power consumption and electromagnetic emission.

You have two main options:

- Switch off the clock and reset the peripheral. You do this using the CFG_PCGR0 and CFG_PCGR0 registers.
- Switch off the clock without resetting the peripheral. In this case, when you switch the clock back on, the peripheral resumes operation in the state prior to the clock switch-off. You control this using the CFG_PCGRB0 and CFG_PCGRB0 registers

Use the following procedure to switch the clock of any peripheral or module on or off:

- After the Reset phase, the peripheral clock is switched-off and the peripheral is kept under reset (RAM is a exception to this, see *Table 13*.)
- To switch on a peripheral clock, set the corresponding bit in the CFG_PCGR[1:0] registers.
- To switch off the clock and reset the peripheral, clear the corresponding bit in the CFG_PCGR[1:0] registers.
- To switch off the clock without resetting the peripheral, set the corresponding bit in the CFG_PCGRB[1:0] registers. To switch the peripheral clock on again, clear the corresponding bit in the CFG_PCGRB[1:0] registers.

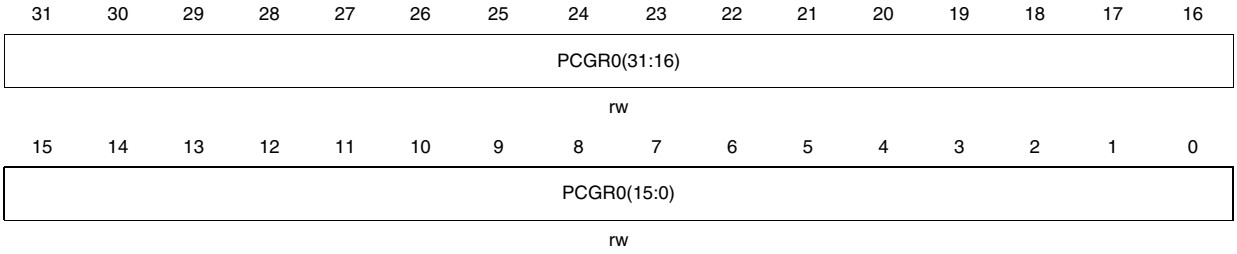
Bits marked as reserved must be left at their reset value.

Note: After enabling a peripheral/port by setting the corresponding bit in the CFG_PCGR[1:0] registers, the clock to the peripheral starts running but, during the first 8 clock cycles, the peripheral itself is kept under reset. You must wait at least this period of time before accessing the peripheral. Any peripheral access during this 8 clock cycle delay will have an unpredictable effect.

4.3.2 Peripheral clock gating register 0 (CFG_PCGR0)

Address Offset: 08h

Reset value: 0000 0001h



Bits 31:0 = **PCGR0[31:0]**: *Peripheral Clock Gating Register 0*.
0: The module is turned off (no clock provided) and kept under reset.
1: The module receives the system clock.
The following table shows which module is mapped to each PCGR0 bit.

Table 13. PCGR0[31:0], PCGBR0[31:0] and PECGR0[31:0] bit mapping and reset Values

Bit	Module	PCGR0 and PCGBR0 Reset Values	PECGR0 Reset Value
0	RAM	1	1
1	I2C 0	0	1
2	WIU	0	1
3	Reserved, must be kept at reset value	0	1
4	UART 0	0	1
5	UART 1	0	1
6	TIM 0	0	1
7	TIM 1	0	1
8	TB 0	0	1
9	Reserved, must be kept at reset value	0	1
10	CAN 0	0	1
11	CAN 1	0	1
12	PWM 0	0	1
13	PWM 1	0	1
14	PWM 2	0	1
15	PWM 3	0	1
16	PWM 4	0	1
17	PWM 5	0	1
18	Port 0	0	1
19	Port 1	0	1
20	Port 2	0	1
21	Port 3	0	1
22	Port 4	0	1
23	Port 5	0	1
24	Port 6	0	1
25	BSPI 0	0	1
26	BSPI 1	0	1
27	BSPI 2	0	1
28	ADC	0	1
29	EIC	0	1
30	Wake-up Timer	0	1
31	Reserved, must be kept at reset value	0	1

4.3.3 Peripheral clock gating register 1 (CFG_PCGR1)

Address Offset: 0Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PCGR1(31:16)															
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCGR1(15:0)															
rw															

Bits 31:0 = **PCGR1[31:0]**: *Peripheral Clock Gating Register 1*.
0: The module is turned off (no clock provided) and kept under reset.
1: The module receives the system clock.
The following table shows which module is mapped to each PCGR1 bit.

Table 14. PCGR1[31:0], PCGBR1[31:0] and PECGR1[31:0] bit mapping and reset values

Bit	Module	PCGR1 and PCGBR1 Reset Values	PECGR1 Reset Value
0	I^2C 1	0	1
1	Reserved, must be kept at reset value	0	1
2	Reserved, must be kept at reset value	0	1
3	TIM 5	0	1
4	TIM 6	0	1
5	TIM 7	0	1
6	TIM 8	0	1
7	TIM 9	0	1
8	UART2	0	1
9	UART3	0	1
10	Reserved, must be kept at reset value	0	1
11	Reserved, must be kept at reset value	0	1
12	Reserved	0	1
13	TB 1	0	1
14	TB 2	0	1
15	Reserved, must be kept at reset value	0	1
16	TIM 2	0	1
17	TIM 3	0	1
18	TIM 4	0	1

Table 14. PCGR1[31:0], PCGBR1[31:0] and PECGR1[31:0] bit mapping and reset values (continued)

Bit	Module	PCGR1 and PCGBR1 Reset Values	PECGR1 Reset Value
19	RTC	0	1
20	DMA 0	0	1
21	DMA 1	0	1
22	DMA 2	0	1
23	DMA 3	0	1
24	Reserved	0	1
25	Reserved	0	1
26	Reserved	0	1
27	Reserved	0	1
28	Reserved	0	1
29	Native Bus Arbiter	0	1
30	AHB Arbiter	0	1
31	Reserved	0	1

4.3.4 Peripheral clock gating register B0 (CFG_PCGRB0)

Address Offset: 18h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PCGBR0(31:16)															
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCGBR0(15:0)															
rw															

Bits 31:0 = **PCGBR0[31:0]**: *Peripheral Clock Gating Register B0*.
0: The module behavior is defined by the corresponding PCGR0 bit configuration.
1: If the corresponding bit of PCGR0 is set it switches off the clock of the module.
The system modules are mapped in the same bits as register CFG_PCGR0. Please refer to *Table 13 on page 58*.

4.3.5 Peripheral clock gating register B1 (CFG_PCGRB1)

Address Offset: 1Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PCGBR1(31:16)															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCGBR1(15:0)															
rw															

Bits 31:0 = **PCGBR1[31:0]**: *Peripheral Clock Gating Register B1*.
0: The module behavior is defined by the corresponding PCGR1 bit configuration.
1: If the corresponding bit of PCGR1 is set it switches off the clock of the module.
The system modules are mapped in the same bits as register CFG_PCGR1. Please refer to *Table 14 on page 59*.

4.3.6 TIM external clock select register (CFG_TIMSR)

Address Offset: 20h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED						TIMECKS(9:0)									
rw						rw									

This register allows to select the external input clock of each TIMn (n=9:0) module between the following two different sources:

- Reference clock f_{EXT} (see *Figure 13 on page 42*). All TIMn (n=0:9) modules receive the same f_{EXT} clock signal on their EXTCLK input (see *Figure 33 on page 159*);
- Timer ICAPAn (n=0:9) input pin. TIMn module receives ICAPAn signal on its EXTCLK input.

Bits 31:10 = Reserved, must be kept at reset value.

Bits 9:0 = **TIMECKS[9:0]**: *TIM[9:0] External Clock Select*.
0: EXTCLK input pin of TIMn module is connected to f_{EXT}.
1: EXTCLK input pin of TIMn module is connected to the corresponding ICAPAn input pin.

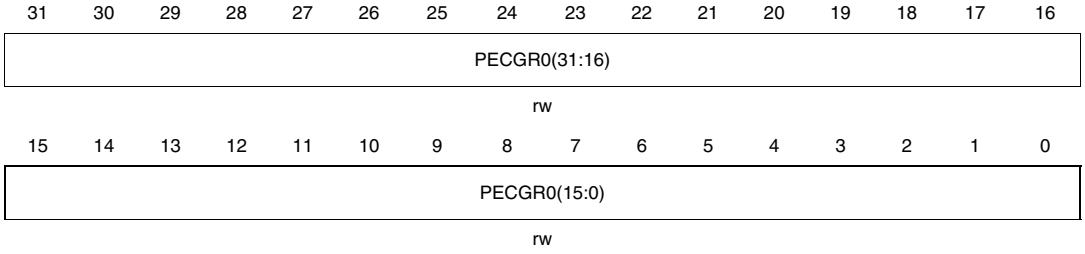
4.3.7 Clock management in emulation mode

You can stop the clock of a specified peripheral when the ARM7TDMI enters emulation mode, using the control bits in the PECGR[1:0] registers.

4.3.8 Peripheral emulation clock gating register 0 (CFG_PECGR0)

Address Offset: 10h

Reset value: FFFF FFFFh



The bits marked as reserved must be kept at 1 (reset value).

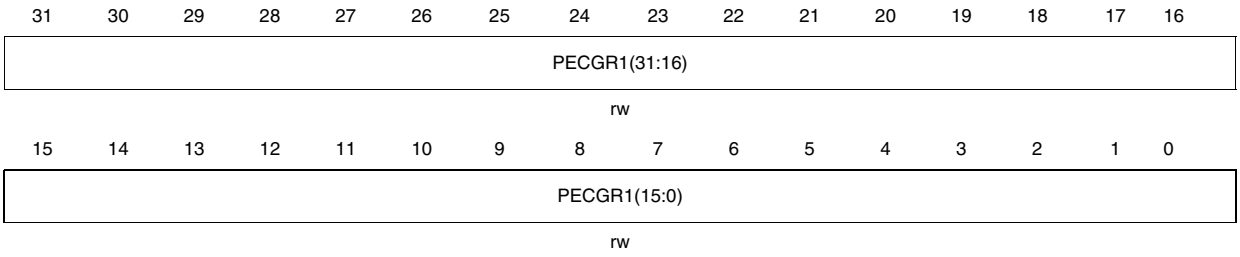
Bits 31:0 = **PECGR0[31:0]**: *Peripheral Emulation Clock Gating Register 0*.
0: The peripheral clock is stretched when ARM7TDMI enters emulation mode, stopping the peripheral.
1: The peripheral clock is enabled when ARM7TDMI enters emulation mode, allowing peripheral operation.

The system modules are mapped in the same bits as register PCGR0. Please refer to *Table 13 on page 58*.

4.3.9 Peripheral emulation clock gating register 1 (CFG_PECGR1)

Address Offset: 14h

Reset value: FFFF FFFFh



The bits marked as Reserved must be kept at 1 (reset value).

Bits 31:0 = **PECGR1(31:0)**: *Peripheral Emulation Clock Gating Register 1*.
0: The peripheral clock is stretched when the ARM7TDMI enters emulation mode, stopping the peripheral from running.
1: The peripheral clock is enabled when the ARM7TDMI enters emulation mode, allowing the peripheral to continue operating.

The system modules are mapped in the same bits as register PCGR1. Please refer to *Table 14 on page 59*.

4.4 BSPI and UART management in emulation mode

The BSPI and UART peripherals have to be managed with particular care when the system enters emulation mode.

In emulation mode, a read operation of the RX FIFO of the above modules pops a received frame from the FIFO. As a result, when emulation mode is exited, the popped frames are no longer available to the application and the control flags related to the RX FIFO may be changed.

A special register (CFG_ESPR) allows you to protect the FIFO structure when a UART or BSPI read operation is performed in emulation mode.

4.4.1 Emulation serial protection register (CFG_ESPR)

Address Offset: 2Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	BSPI2	BSPI1	BSPI0	UART3	UART2	UART1	UART0
									rw	rw	rw	rw	rw	rw	rw

Bits 31:7 = Reserved, must be written to ‘0’.

Bits 6:4 = **BSPI[2:0]**: *BSPI[2:0] configuration in emulation mode*.
These bits can be set to protect the BSPI[2:0] modules when a read operation is performed in emulation mode. The read operation is considered *protected* if it does not modify the BSPI internal FIFO structure.
For each bit n (n=0,2):
0: BSPIn is not protected when ARM7TDMI enters in emulation mode.
1: BSPIn is protected when ARM7TDMI enters in emulation mode.

Bits 3:2 = Reserved, always return ‘0’ when read.

Bits 1:0 = **UART[3:0]**: *UART[3:0] configuration in emulation mode*.
These bits can be set to protect the UART[3:0] modules when a read operation is performed in emulation mode. The read operation is considered *protected* if it does not modify the UART internal FIFO structure.
For each bit n (n=0,1):
0: UARTn is not protected when ARM7TDMI enters emulation mode.
1: UARTn is protected when ARM7TDMI enters emulation mode.

4.5 CFG register map

Table 15. CFG register map

Addr. offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	CFG_R0	reserved																				T_MODE	BOOT	USER1	USER2	JTBT	res.	DMABSPI1	DMABSPI0	reserved			REMAP
04h	CFG_EITE0	reserved																EITE0(15:0)															
08h	CFG_PCGR0	PCGR0[31:0]																															
0Ch	CFG_PCGR1	PCGR1[31:0]																															
10h	CFG_PECGR0	PECGR0[31:0]																															
14h	CFG_PECGR1	PECGR1[31:0]																															
18h	CFG_PCGRB0	PCGBR0[31:0]																															
1Ch	CFG_PCGRB1	PCGBR1[31:0]																															
20h	CFG_TIMSR	reserved																						TIMECKS[9:0]									
24h	CFG_EITE1	reserved																EITE1[15:0]															
28h	CFG_EITE2	reserved																EITE2(15:0)															
2Ch	CFG_ESPR	reserved																								BSPI2	BSPI1	BSPI0	UART3	UART2	UART1	UART0	
30h	CFG_R1	reserved																							FLPOD				WUP0S	EIFILT	LP VRC C [1:0]		
34h	CFG_DIDR	DIDR[31:0]																															

See *Table 1 on page 17* for the base address

5 Clock tree map

This section summarizes the clock paths and the possible configurations for the various STR73x modules.

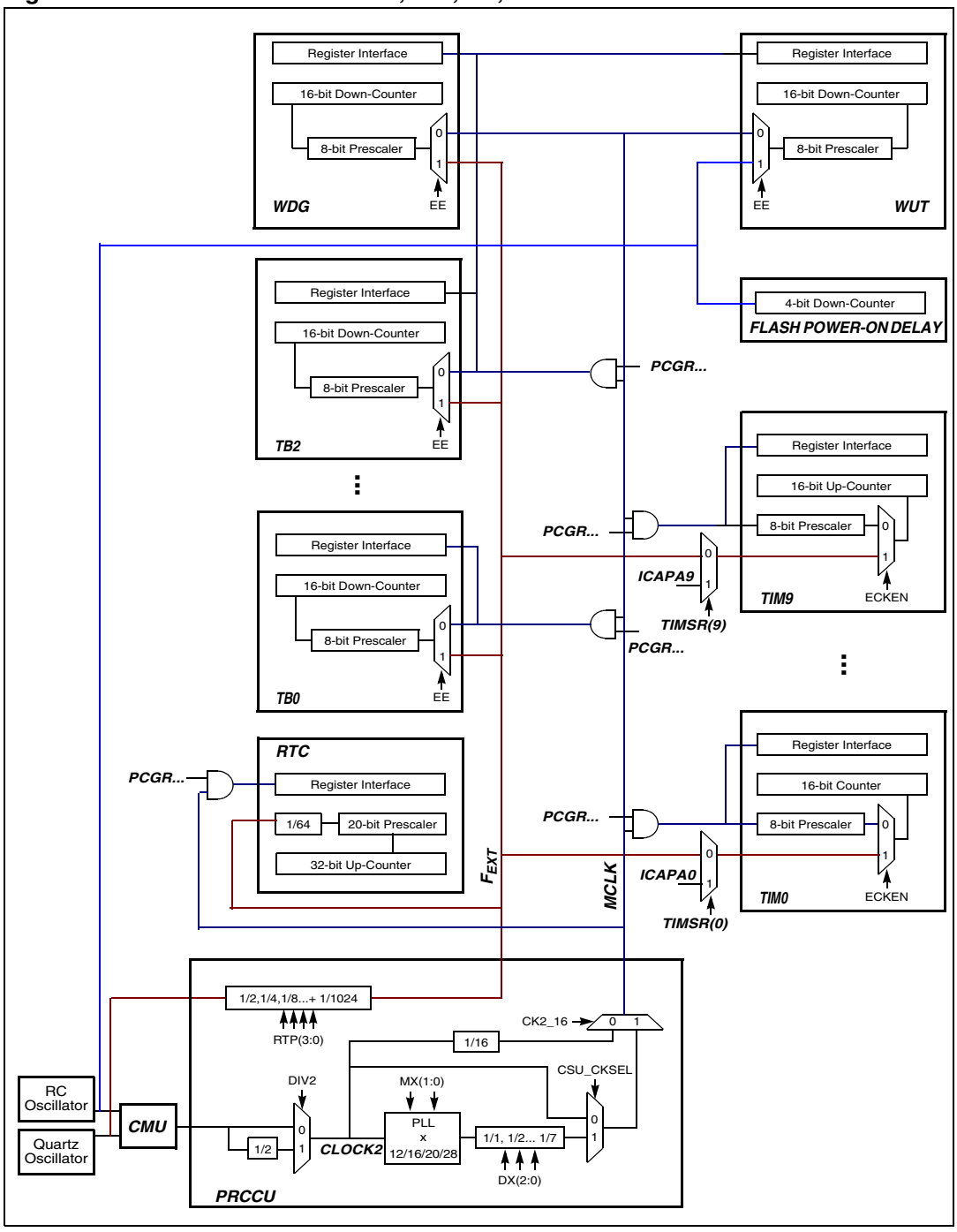
Figure 14 on page 66 show different modules (Wake-up Timer, Watchdog, TIM,TB, Real Time Clock) and how different clock signals are used to feed the counters. The PRCCU is also shown in the diagram since it generates and distributes the clock signals to the various modules.

The advantage of choosing the reference clock for the counters is that it can be used to provide a time basis independent from the PLL clock frequency (system clock) to guarantee regular time frames, without any scaling effects when the system is put into low power mode or configures with a slower system clock signal.

The internal RC-Oscillator feeds the Wake-up Timer directly and the circuit is used to manage the FLASH Power-on delay for both LPWFI and STOP mode exit events (please refer to *Section 4.1*).

All the other modules, not explicitly mentioned in this section, work with the system clock (MCLK) optionally prescaled through the internal frequency dividers and gated by the PCGR0/PCGRB0 and PCGR1/PCGRB1 registers.

Figure 14. Clock distribution: RTC, TIM, TB, WDG and WUT modules.



6 I/O ports

6.1 Functional description

Each of the General Purpose I/O Ports has three 16-bit configuration registers (PC0, PC1, PC2) and one 16-bit Data register (PD).

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, you can configure each port bit individually as input, output, alternate function etc.

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 16-bit words. Byte or bit-wise access is not allowed.

Figure 15 shows the basic structure of an I/O Port bit.

Figure 15. Basic structure of an I/O port bit

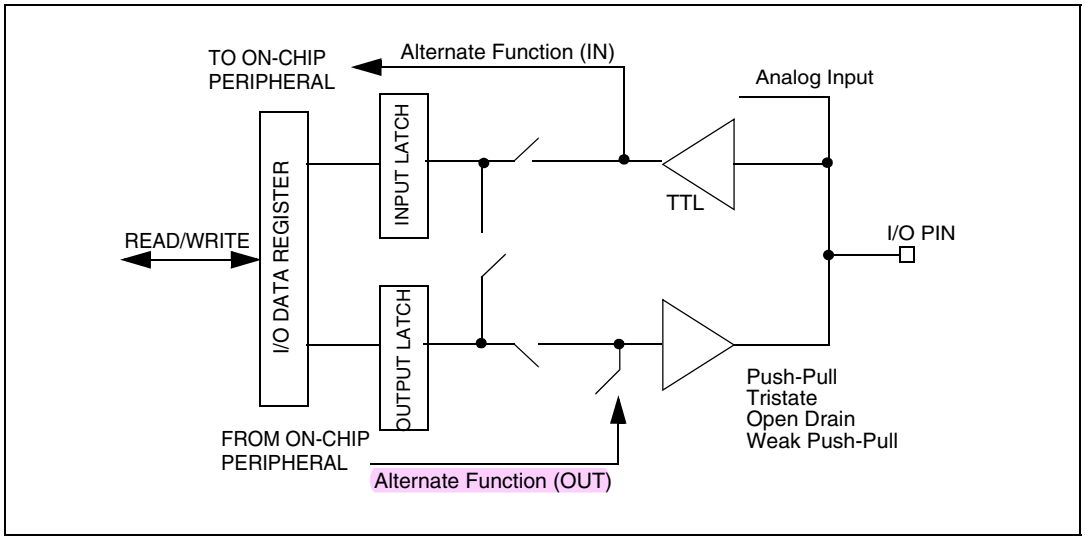


Table 16. Port bit configuration table

Port Configuration Registers (bit)	Values							
PC0(n)	0	1	0	1	0	1	0	1
PC1(n)	0	0	1	1	0	0	1	1
PC2(n)	0	0	0	0	1	1	1	1
Configuration	HiZ/AIN	IN	reserved	IPUPD	OUT	OUT	AF	AF
Output	TRI	TRI		WP	OD	PP	OD	PP
Input	-	TTL		TTL	TTL	TTL	TTL	TTL

Notes:
AF: Alternate Function
AIN: Analog Input
HiZ: High impedance
IN: Input
IPUPD: Input Pull Up /Pull Down
OD: Open Drain
OUT: Output
PP: Push-Pull
TRI: Tristate
TTL: TTL Input levels
WP (1): Weak Push-Pull

1. Depending on PD(n) value it behaves as Weak Pull-up (PD=1) or Weak Pull-down (PD=0)

6.1.1 General purpose I/O (GPIO)

At reset the I/O ports are configured as general purpose (memory mapped I/O).

When you write to the I/O Data register the data is always loaded in the Output Latch. The Output Latch holds the data to be output while the Input Latch captures the data present on the I/O pin.

A read access to the I/O Data register reads the Input Latch or the Output Latch depending on whether the Port bit is configured as input or output.

6.1.2 Alternate function I/O (AF)

The alternate functions for each pin are listed in the datasheet. If you configure a port bit as Alternate Function, this disconnects the output latch and connects the pin to the output signal of an on-chip peripheral.

To use the alternate function, you also have to enable it in the peripheral control registers. Only one alternate function can be used on each pin

- For alternate function inputs, the port must be configured in Input mode (floating, pull-up or pull-down) and the input pin must be driven externally.

Note: It is also possible to emulate by software the AFI input pin by programming the GPIO controller. In this case, the port should be configured in Alternate Function Output mode. And obviously, the corresponding port should not be driven externally as it will be driven by the software using the GPIO controller.

- For AF output or input-output, the port bit must be in AF configuration

External Interrupts/wake-up lines

Some ports have external interrupt capability (see datasheet). To use external interrupts, the port must be configured in input mode. For more information on interrupts and wake-up lines, refer to *Section 7*.

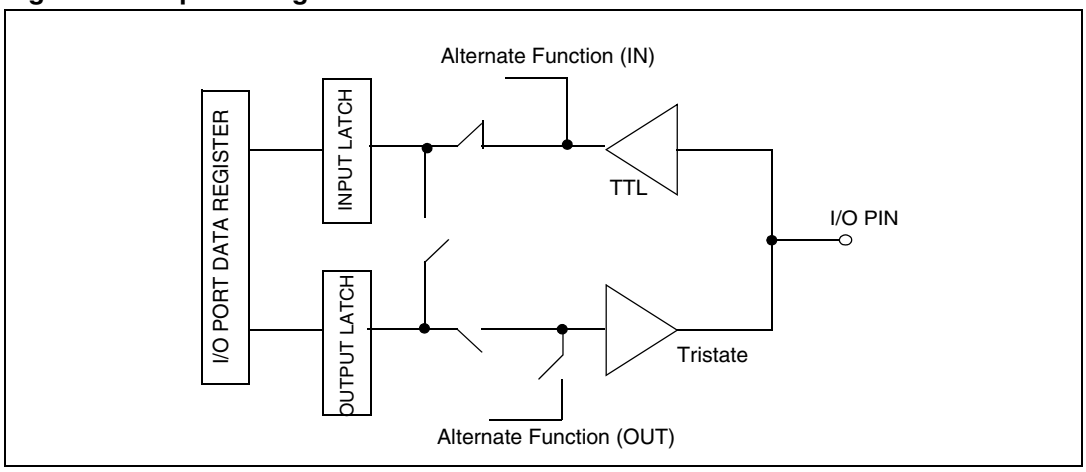
6.1.3 Input configuration

When the I/O Port is programmed as Input:

- The Output Buffer is forced tristate
- The data present on the I/O pin is sampled into the Input Latch every clock cycle
- A read access to the Data register gets the value in the Input Latch.

The *Figure 16* shows the Input Configuration of the I/O Port bit.

Figure 16. Input configuration



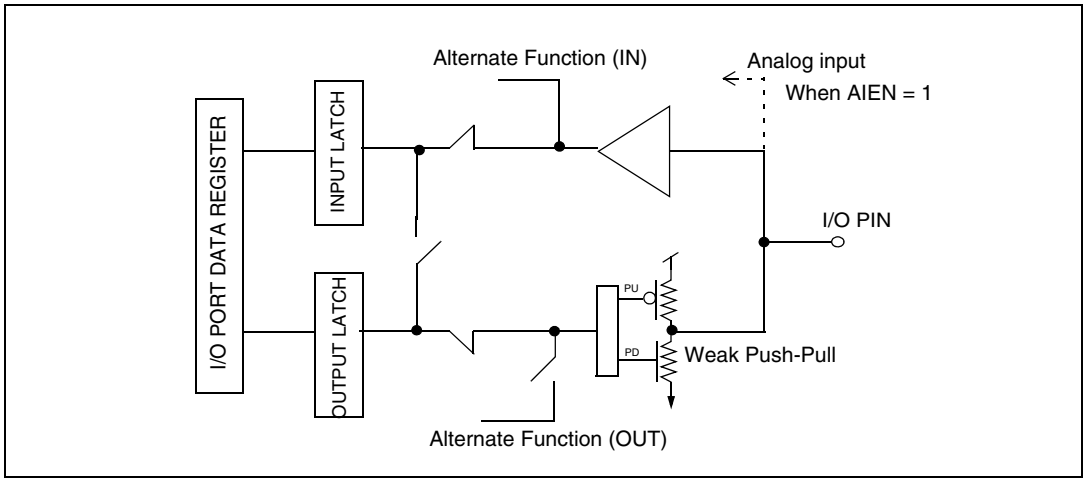
6.1.4 Input pull up/pull down configuration

When the I/O Port is programmed as Input Pull Up/Pull Down:

- The Output Buffer is turned on in Weak Push-Pull configuration and software can write the appropriate level in the output latch to activate the weak pull-up or pull-down as required.
- The data in the Output Latch drives the I/O pin (a logic zero activates a weak pull-down, a logic one activates a weak pull-up).
- A read access to the I/O Data register gets the Input Latch value.

The *Figure 17* shows the Input PUPD Configuration of the I/O Port.

Figure 17. Input Pull Up/Pull Down configuration



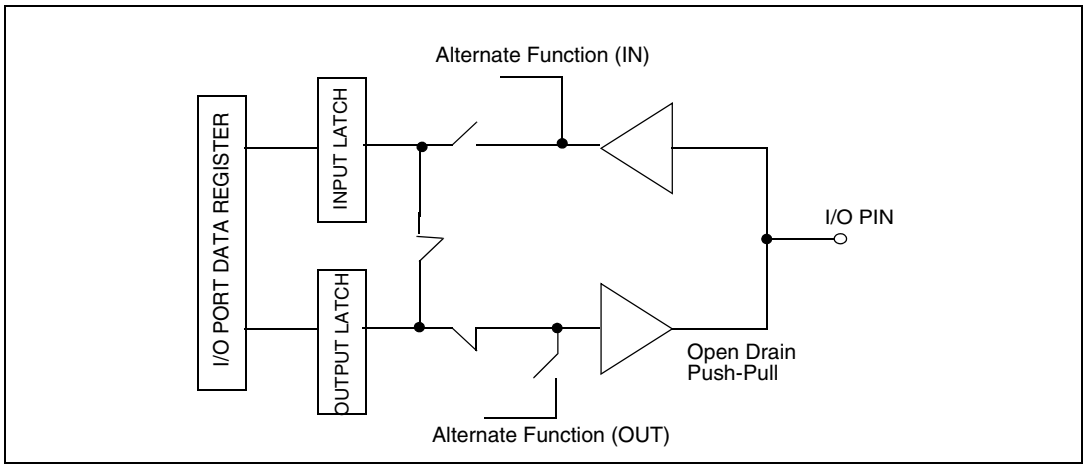
6.1.5 Output configuration

When the I/O Port is programmed as Output:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The data in the Output Latch drives the I/O pin
- A read access to the I/O Data register gets the Output Latch value.

The *Figure 18 on page 70* shows the Output Configuration of the I/O Port bit.

Figure 18. Output configuration



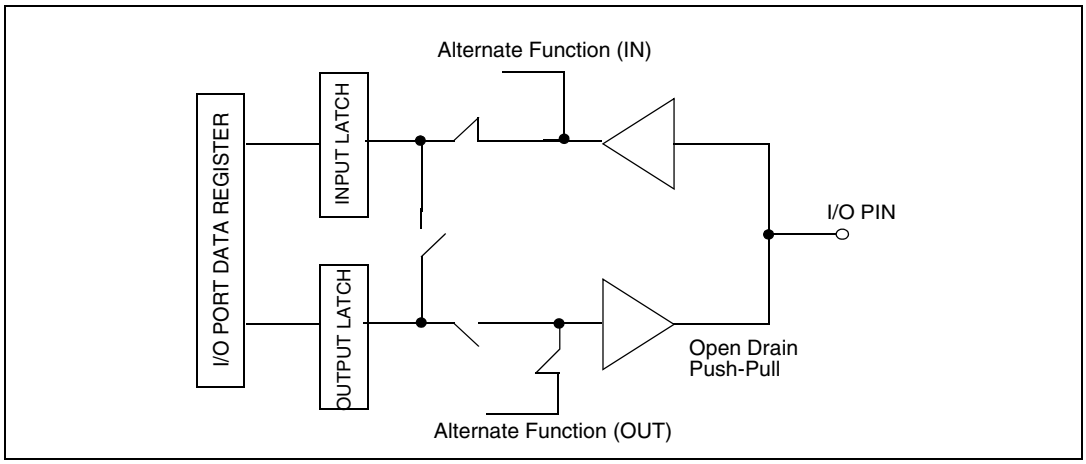
6.1.6 Alternate Function configuration

When the I/O Port is programmed as Alternate Function:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The Output Buffer is driven by the signal coming from the peripheral (alternate function out)
- The data present on the I/O pin is sampled into the Input Latch every clock cycle
- A read access to the Data register gets the value in the Input Latch.

The *Figure 19* shows the Alternate Function Configuration of the I/O Port bit.

Figure 19. Alternate Function configuration



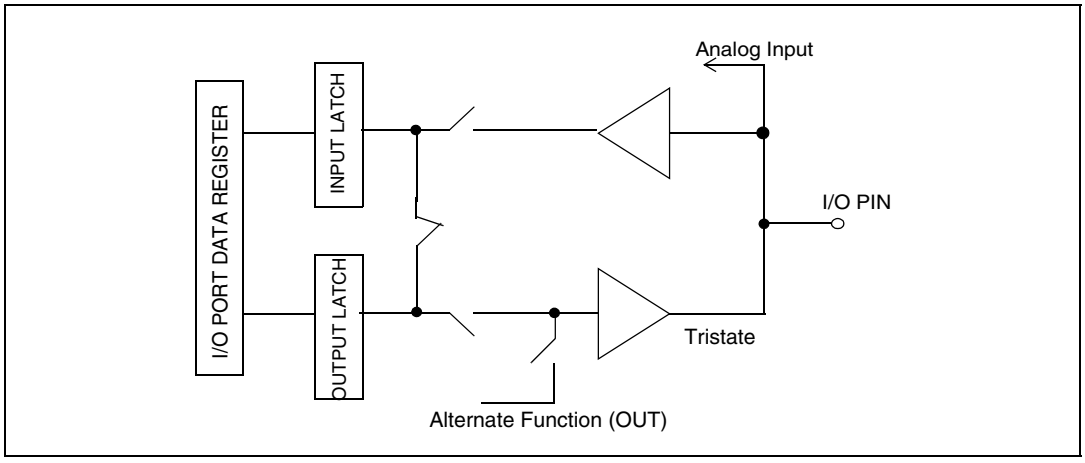
6.1.7 High impedance-analog input configuration

When the I/O Port is programmed as High impedance-Analog Input Configuration:

- The Output Buffer is forced tristate
- The Input Buffer is disabled (the Alternate Function Input is forced to a constant value)
- The Analog Input can be input to an Analog peripheral
- A read access to the I/O Data register gets the Output Latch value

Figure 20 shows the high impedance-analog input configuration of the I/O Port bit.

Figure 20. High impedance-analog input configuration



6.2 Register description

The I/O port registers cannot be accessed by byte.

Port Configuration Register 0 (PC0)

Address Offset: 00h

Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C015	C014	C013	C012	C011	C010	C09	C08	C07	C06	C05	C04	C03	C02	C01	C00
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **C0[15:0]:** *Port Configuration bits*
See *Table 16 on page 68* to configure the I/O Port.

Port configuration register 1 (PC1)

Address Offset: 04h

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C115	C114	C113	C112	C111	C110	C19	C18	C17	C16	C15	C14	C13	C12	C11	C10
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **C1[15:0]:** *Port Configuration bits*
See *Table 16 on page 68* to configure the I/O Port.

Port configuration register 2 (PC2)

Address Offset: 08h

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C215	C214	C213	C212	C211	C210	C29	C28	C27	C26	C25	C24	C23	C22	C21	C20
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **C2[15:0]:** *Port Configuration bits*
See *Table 16 on page 68* to configure the I/O Port.

I/O data register (PD)

Address Offset: 0Ch

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **D[15:0]:** *I/O Data bits*
A writing access to this register always writes the data in the Output Latch.
A reading access reads the data from the Input Latch in Input and Alternate function configurations or from the Output Latch in Output and High impedance configurations.

Refer to the datasheet for the device pin description.

6.2.1 I/O Port register map

The following table summarizes the registers implemented in each I/O port.

Table 17. I/O post register map

Addr. offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	PC0	C0[15:0]															
04h	PC1	C1[15:0]															
08h	PC2	C2[15:0]															
0Ch	PD	D[15:0]															

See *Table 2 on page 17* for base address

7 Interrupts

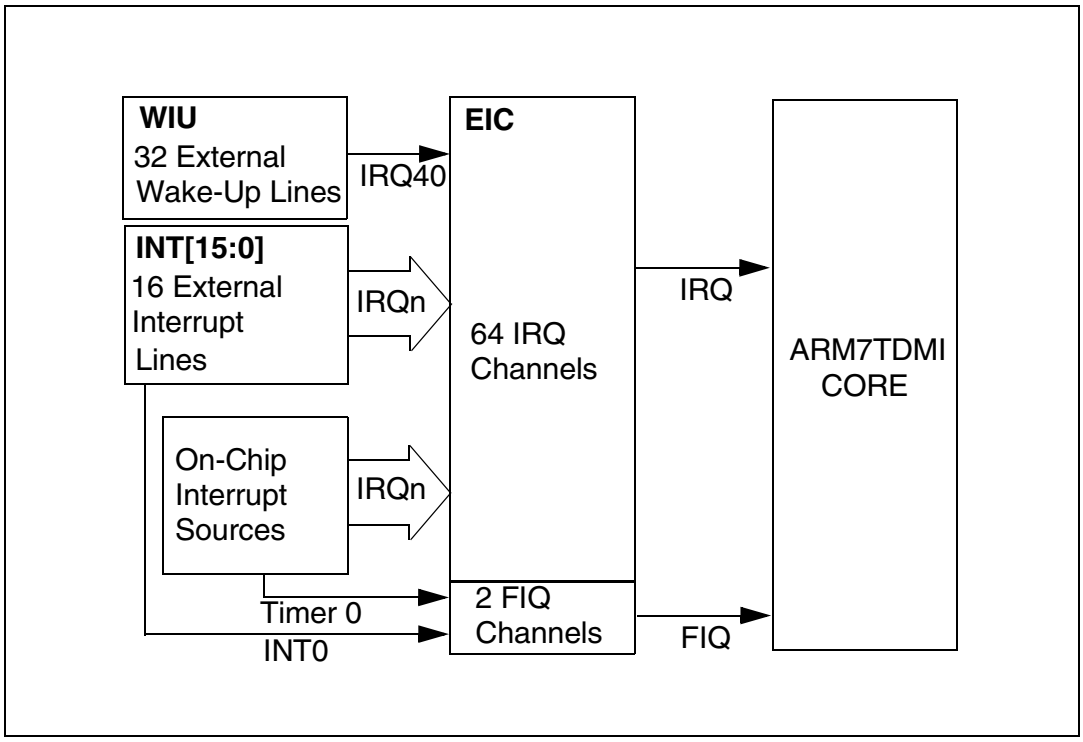
The ARM7 CPU provides two levels of interrupt, FIQ (Fast Interrupt Request) for fast, low latency interrupt handling and IRQ (Interrupt Request) for more general interrupts.

The Enhanced Interrupt Controller manages interrupt requests from three sources:

- On-chip peripherals and modules
- External interrupts
- WIU Wake-up lines

Refer to *Figure 21*.

Figure 21. Interrupt management overview

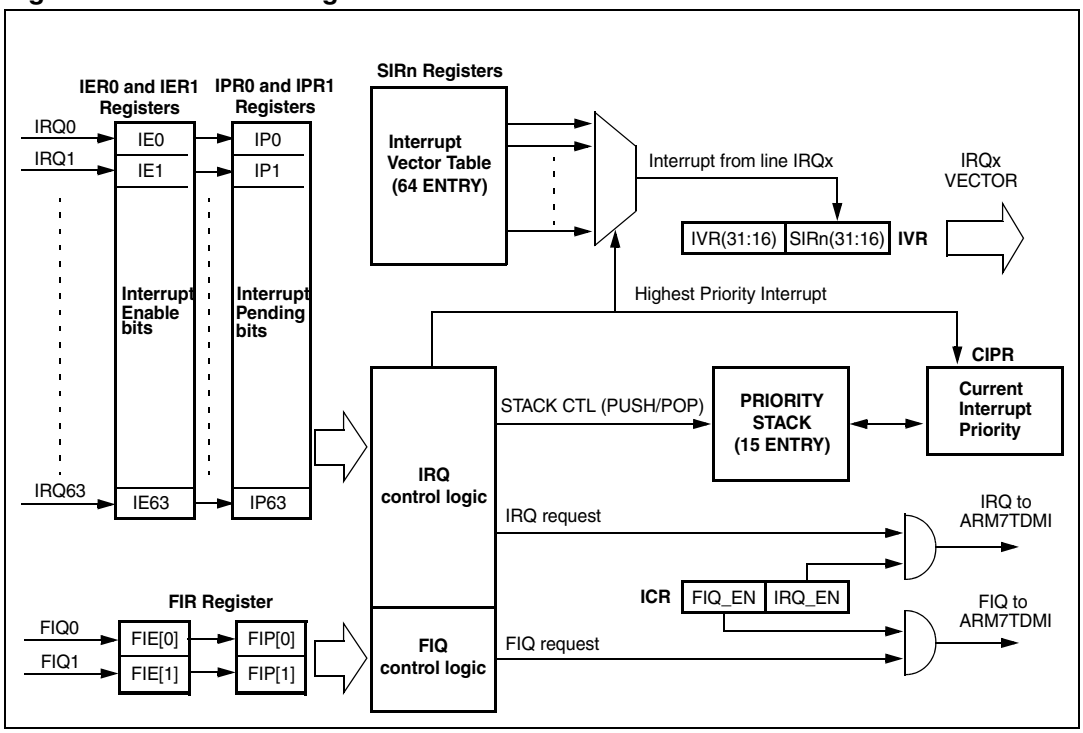


7.1 Enhanced interrupt controller (EIC)

The ARM7TDMI CPU provides two levels of interrupt, FIQ (Fast Interrupt Request) for fast, low latency interrupt handling and IRQ (Interrupt Request) for more general interrupts.

Hardware handling of multiple interrupt channels, interrupt priority and automatic vectorization require therefore a separate Enhanced Interrupt Controller (EIC).

Figure 22. EIC block diagram



The Enhanced Interrupt Controller (EIC) performs hardware handling of multiple interrupt channels, interrupt priority arbitration and vectorization. It provides:

- 64 maskable interrupt channels, mapped on the IRQ interrupt request line of the ARM CPU
- 16 programmable priority levels for each interrupt channel mapped on IRQ (15 = highest priority, 1= lowest priority, 0 = never served)
- Hardware support for interrupt nesting (up to 15 interrupt requests can be nested), with internal hardware nesting stack
- 2 maskable interrupt channels, mapped on FIQ interrupt request line of the ARM CPU, with neither priority nor vectorization
- 16 external interrupts INT [15:0]
- 32 external wake-up lines from the WIU block are mapped on IRQ40.

The EIC performs the following operations without software intervention:

- Rejects/accepts an interrupt request according to the related channel mask bit
- Compares all pending IRQ requests with the current priority level. The IRQ is asserted to the ARM7 if the priority of the current interrupt request is higher than the stored current priority
- Loads the address vector of the highest priority IRQ to the Interrupt Vector Register
- Saves the previous interrupt priority in the HW priority stack whenever a new IRQ is accepted
- Updates the Current Interrupt Priority Register with the new priority whenever a new interrupt is accepted

7.1.1 IRQ interrupt vector table

Up to 64 interrupt channels are mapped on low priority ARM7TDMI interrupt request pin (IRQ). If multiple interrupt sources are mapped on the same interrupt vector, software has read the peripheral interrupt flag register to determine the exact source of interrupt (see Interrupt Flags column in *Table 18*).

Table 18. IRQ Interrupt vector table

Vector	Acronym	Peripheral interrupt	Peripheral interrupt flags
IRQ0	IRQ.CMU_PRCCU	Clock Monitor Unit (CMU) / Power Reset and Clock Control unit (PRCCU)	4 (CMU_STAT), 2 (PRCCU_CFR)
IRQ1	IRQ.INT1	External Interrupt INT1	
IRQ2	IRQ.INT2	External Interrupt INT2	
IRQ3	IRQ.INT3	External Interrupt INT3	
IRQ4	IRQ.INT4	External Interrupt INT4	
IRQ5	IRQ.INT5	External Interrupt INT5	
IRQ6	IRQ.INT6	External Interrupt INT6	
IRQ7	IRQ.INT7	External Interrupt INT7	
IRQ8	IRQ.INT8	External Interrupt INT8	
IRQ9	IRQ.INT9	External Interrupt INT9	
IRQ10	IRQ.INT10	External Interrupt INT10	
IRQ11	IRQ.INT11	External Interrupt INT11	
IRQ12	IRQ.INT12	External Interrupt INT12	
IRQ13	IRQ.INT13	External Interrupt INT13	
IRQ14	IRQ.INT14	External Interrupt INT14	
IRQ15	IRQ.INT15	External Interrupt INT15	
IRQ16	IRQ.DMA	DMA Transfer Error	1 (ARB_CTLR)
IRQ17	IRQ.TIM1	Timer 1 global interrupt	5 (TIM1_SR)
IRQ18	IRQ.TIM2	Timer 2 global interrupt	5 (TIM2_SR)

Table 18. IRQ Interrupt vector table (continued)

Vector	Acronym	Peripheral interrupt	Peripheral interrupt flags
IRQ19	IRQ.TIM3	Timer 3 global interrupt	5 (TIM3_SR)
IRQ20	IRQ.TIM4	Timer 4 global interrupt	5 (TIM4_SR)
IRQ21	IRQ.TBU0	Timebase Unit 0 End of Count Interrupt	1 (TBU0_SR)
IRQ22	IRQ.TBU1	Timebase Unit 1 End of Count Interrupt	1 (TBU1_SR)
IRQ23	IRQ.TBU2	Timebase Unit 2 End of Count Interrupt	1 (TBU2_SR)
IRQ24	IRQ.TIM5	Timer 5 global interrupt	5 (TIM5_SR)
IRQ25	IRQ.TIM6	Timer 6 global interrupt	5 (TIM6_SR)
IRQ26	IRQ.TIM7	Timer 7 global interrupt	5 (TIM7_SR)
IRQ27	IRQ.TIM8	Timer 8 global interrupt	5 (TIM8_SR)
IRQ28	IRQ.TIM9	Timer 9 global interrupt	5 (TIM9_SR)
IRQ29	Reserved		
IRQ30	Reserved		
IRQ31	IRQ.UART2	UART 2 global interrupt	9 (UART2_SR)
IRQ32	IRQ.UART3	UART 3 global interrupt	9 (UART3_SR)
IRQ33	IRQ.FLASH	Flash End of Write	1 (FLASH_CR0)
IRQ34	IRQ.PWM0	PWM 0 Compare Period Interrupt	1 (PWM0_CPI)
IRQ35	IRQ.PWM1	PWM 1 Compare Period Interrupt	1 (PWM1_CPI)
IRQ36	IRQ.PWM2	PWM 2 Compare Period Interrupt	1 (PWM2_CPI)
IRQ37	IRQ.PWM3	PWM 3 Compare Period Interrupt	1 (PWM3_CPI)
IRQ38	IRQ.PWM4	PWM 4 Compare Period Interrupt	1 (PWM4_CPI)
IRQ39	IRQ.PWM5	PWM 5 Compare Period Interrupt	1 (PWM5_CPI)
IRQ40	IRQ.WIU	Wake-Up Unit Global Interrupt	32 (WIU_PR)
IRQ41	IRQ.WDG_WUT	Watchdog Timer (WDG)/Wake-Up Timer (WUT) End of Count	1 (WDG_SR), 1 (WUT_SR)
IRQ42	IRQ.BSPI0	BSPI 0 global interrupt	5 (BSPI0_CSR2)
IRQ43	IRQ.BSPI1	BSPI 1 global interrupt	5 (BSPI1_CSR2)
IRQ44	IRQ.BSPI2	BSPI 2 global interrupt	5 (BSPI2_CSR2)
IRQ45	IRQ.UART0	UART 0 global interrupt	9 (UART0_SR)
IRQ46	IRQ.UART1	UART 1 global interrupt	9 (UART0_SR)
IRQ47	IRQ.I2C0_ITERR	I2C 0 general interrupt	10 (I2C0_SR1, SR2)
IRQ48	IRQ.I2C1_ITERR	I2C 1 general interrupt	10 (I2C0_SR1, SR2)

Table 18. IRQ Interrupt vector table (continued)

Vector	Acronym	Peripheral interrupt	Peripheral interrupt flags
IRQ49	Reserved		
IRQ50	Reserved		
IRQ51	IRQ.I2C0_ITDDC	I2C 0 general/DMA interrupt	10 (I2C0_SR1, SR2)
IRQ52	IRQ.I2C1_ITDDC	I2C 1 general/DMA interrupt	10 (I2C0_SR1, SR2)
IRQ53	Reserved		
IRQ54	Reserved		
IRQ55	IRQ.CAN0	CAN 0 global interrupt	32 (CAN0_IP1R, CAN0_IP2R)
IRQ56	IRQ.CAN1	CAN 1 global interrupt	32 (CAN1_IP1R, CAN1_IP2R)
IRQ57	IRQ.CAN2	CAN 2 global interrupt	32 (CAN2_IP1R, CAN2_IP2R)
IRQ58	IRQ.DMA0	DMA 0 global interrupt	8 (DMA0_SR)
IRQ59	IRQ.DMA1	DMA 1 global interrupt	8 (DMA1_SR)
IRQ60	IRQ.DMA2	DMA 2 global interrupt	8 (DMA2_SR)
IRQ61	IRQ.DMA3	DMA 3 global interrupt	8 (DMA3_SR)
IRQ62	IRQ.ADC	A/D Converter global interrupt	12 (ADC_PBR)
IRQ63	IRQ.RTC	RTC global interrupt	4 (RTC_CRL

7.1.2 FIQ interrupt vector table

Two maskable interrupt sources are mapped on FIQ vectors, as shown in *Table 19*:

Table 19. FIQ vector table

Vector	Interrupt source
FIQ0	INT0 - External Interrupt 0
FIQ1	TIM0 - Timer 0 Global Interrupt

These sources are also available as normal IRQs. In most cases, you should only enable one FIQ source in your application. If you enable both FIQ sources, then you can determine the source of the interrupt by reading the FIQ pending bits in the EIC register. Bear in mind that FIQ has no priority mechanism, so if simultaneous FIQ events occur, software has to manage the priority by polling the pending bits and managing the concurrency.

7.1.3 IRQ interrupt structure

The EIC (Enhanced Interrupt Controller) implements an interrupt structure pointing the processor to the first instruction location of the channel-specific Interrupt (IRQ) Service Routine.

IVR (Interrupt Vector Register) is the EIC's 32-bit register at address 0xFFFF FC18h acting as pointer. It is composed of two main fields: the upper half word (16 bit) is directly programmable, while the lower half word is the mirrored entry of a register table (named SIR) indexed by the interrupt channel.

The absolute address 0x0000 0018h is where the ARM7TDMI CPU jumps as consequence of an interrupt request on its IRQ line.

The STR73x considers the ARM's 0x0000 0018h location and the EIC's IVR location (0xFFFF FC18h) as distinct addresses. The EIC IVR register must contain the absolute address of the interrupt service routine. The absolute 0x0000 0018h location must contain an instruction which jumps to the location pointed to by the IVR register.

For instance, the absolute location 0x0000 0018h should contain the following instruction:

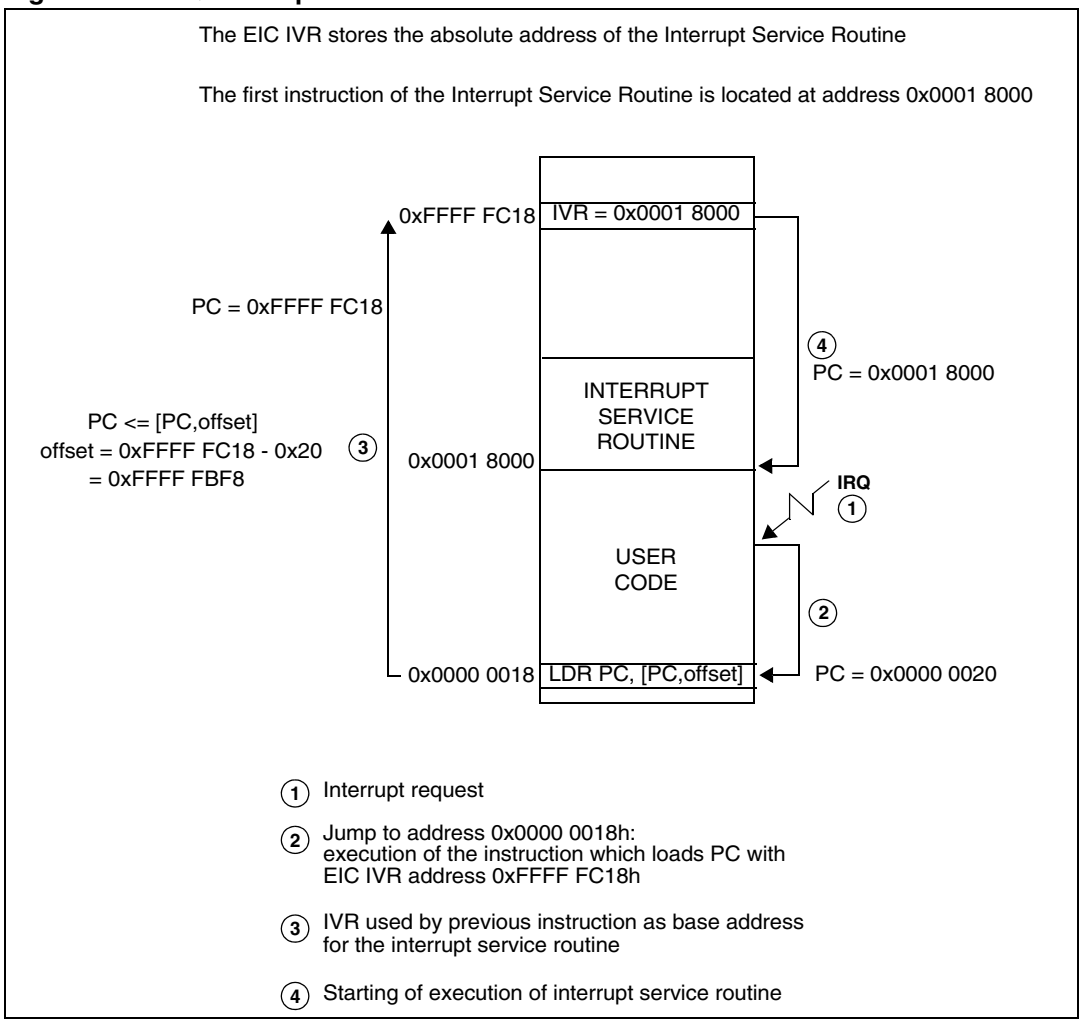
LDR PC, [PC, offset]

where "offset" is what to add to the PC to obtain the EIC IVR address. The instruction above allows the CPU to load the Program Counter register with the address kept in the EIC IVR and so to jump to the location pointed to by the EIC_IVR register.

The user has to consider that when the absolute address 0x0000 0018h is being fetched, the PC is equal to 0x0000 0020h (18h + 8h). So, the offset is equal to:

offset = IVR address – 0x0000 0020h = 0xFFFF FC18h – 0x0000 0020h = 0xFFFF FBF8h

Figure 23. IRQ interrupt structure



This mechanism allows a jump to virtually any location in the 4GB memory space. However, since the channel dependent portion of the IVR is the lower 16 bits, once the base address (in the upper part) has been fixed, the interrupt handler routines can only be within a 64Kbyte range from that base address.

In case interrupt service routines need to be available during FLASH content updating, RAM must be used to store the related code. In this case, the RAM remapping feature can be used: in this way, the address 0x0000 0018h becomes available in RAM memory space, and the above mechanism works similarly.

7.1.4 Priority decoder

The priority decoder is a combinational block continuously calculating the pending channel with the highest IRQ priority. If there is a winner, it updates the IVR (Interrupt Vector Register) with the address of the IRQ interrupt routine that has won the arbitration, and asserts the nIRQ internal signal low. The nIRQ internal signal ORed with the inverted EIC IRQ enable bit (IRQ_EN) corresponds to the ARM7TDMI nIRQ signal.

Each channel has a 4-bit field, the SIPL (Source Interrupt Priority Level) in SIRn (Source Interrupt Register 0-63) defining the channel priority level in the range of 0 (lowest priority) to 15 (highest).

If several channels are assigned with the same priority level, an internal hardware daisy chain fixes the priority between them. The higher the channel address, the higher the priority. If channel 2 and channel 6 are assigned to the same software priority level, and if they are both pending at the same time, channel 6 will be served first.

In order to declare a channel as a winner, the channel must:

- Be pending (EIC_IPR0-1 - Interrupt Pending Register, 64 pending bits, one per channel). In order to be pending, a channel has to be enabled (EIC_IER0-1 - Interrupt Enable Register, 64 enable bits, one per channel). A pending bit will not be set if the corresponding enable bit is not set. If the enable bit is reset while the pending bit is still set, the channel will still be part of the priority decoder until the pending bit is cleared.
- Have the highest priority level, higher than the current one (EIC_CIPR - Current Interrupt Priority Register) and higher than any other pending interrupt channel.
- Have the highest position in the interrupt channel chain if there are multiple pending interrupt channels with the same priority level.

The EIC_CIPR provides the priority of the main program or the priority of the interrupt routine currently being served. At reset, the EIC_CIPR is at 0, it can be modified by software from 0 (lowest) to 15 (highest) for the main program. For an interrupt routine, it can be modified by software from the initialized priority value stored in the EIC_SIRn (Source Interrupt Register 0-63) up to 15. Attempting to write a lower value than the one in EIC_SIRn will have no effect.

For safe operation, it is recommended to disable the global IRQ before modifying the EIC_CIPR, SIR, or IPR registers pending IRQ clearing, to avoid dangerous race conditions. Moreover, if IRQ_EN is cleared in an Interrupt Subroutine, the pending bit related to the IRQ currently being served must not be cleared, otherwise it will no longer be possible to recover the EIC status before popping the stack.

7.1.5 Finite state machine

The Finite State Machine (FSM) has two states, READY and WAIT. The two states correspond to the ARM7TDMI nIRQ line state masked by the EIC global enable bit (IRQ_EN). After reset, the FSM is in READY state (EIC nIRQ line is high). When the priority decoder elects a new winner, the FSM moves from READY to WAIT state and the EIC nIRQ line is asserted low.

To move the FSM back to the ready state, it is mandatory to read the IVR register or to reset the EIC cell. The EIC can be reset by a global reset resetting the entire device or by clearing the EIC bit in the CFG_PCGR0 (Peripheral Clock Gating Register 0).

Reading the IVR always moves the FSM from WAIT to READY state, assuming that the FSM was in WAIT state, and automatically releases the EIC nIRQ line.

There is no flag indicating the FSM state.

7.1.6 Stack

The stack is up to 15 events deep corresponding to the maximum number of nested interrupts. It is used to push and pop the previous EIC state. The data pushed onto the stack

are EIC_CICR (Current Interrupt Channel Register) and EIC_CIPR (Current Interrupt Priority Register).

When the FSM is in WAIT state, reading the IVR raises an internal flag. This pushes the previous CICR and CIPR onto the EIC stack. This happens on the next internal clock cycle after reading the IVR. In the meantime the internal flag is cleared. The EIC_CICR and EIC_CIPR are updated with the value corresponding to the interrupt channel read in IVR.

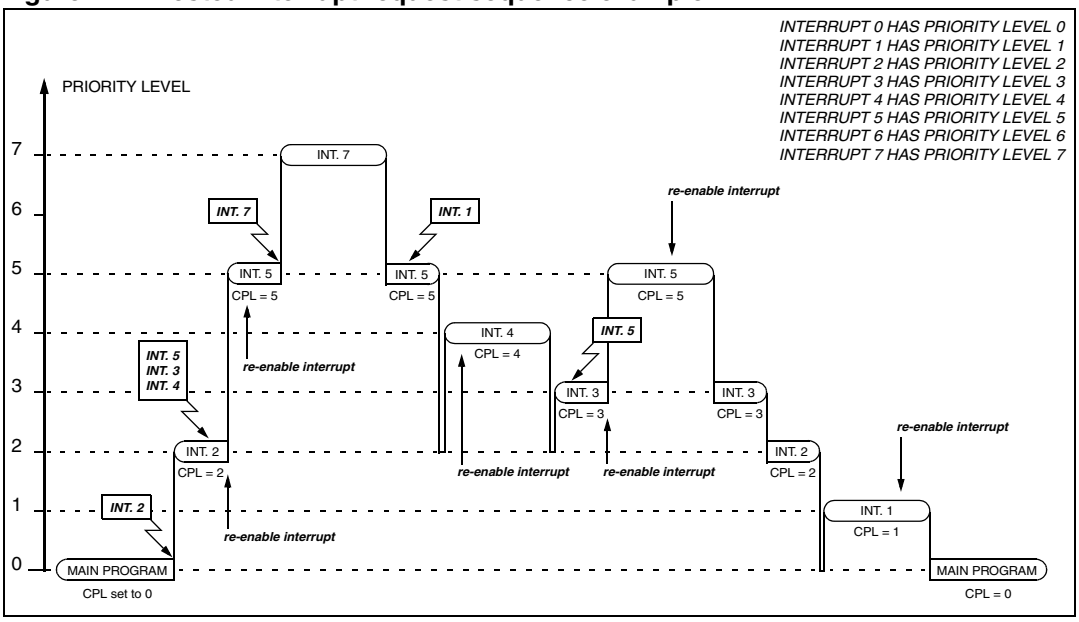
If IVR is read while the FSM is in READY state, the internal flag is not raised and no operation is performed on the EIC internal stack.

A routine can only be interrupted by a event having a higher priority. Consequently, the maximum number of nested interrupts is 15, corresponding to the 15 priority levels, from 1 to 15. An interrupt with priority level 0 can never be executed. In order for the stack to be full, up to 15 interrupts must be nested and each interrupt event must appear sequentially from priority level 1 up to 15. The main program must have priority level 0.

Having all interrupt sources with a priority level 0 could be useful in applications that only use polling.

To pop the stack, the EIC pending bit corresponding to the interrupt in the EIC_CICR register has to be cleared. Clearing any other pending bits will not pop the stack. Take care not to clear a pending bit corresponding to an event still in the stack, otherwise it will not be possible to pop the stack when reaching this stack stage. When the stack is popped, the EIC_CICR and EIC_CIPR are restored with the values corresponding to the previous interrupt event.

Figure 24. Nested interrupt request sequence example



7.1.7 EIC interrupt vectoring

When the ARM7TDMI decodes an IRQ interrupt request, the instruction at the address 0x18 is executed. By this time the IVR register is updated with the address of the highest pending interrupt routine. In order to get the maximum advantage from the EIC mechanism, the instruction at address 0x18 can load the program counter with the address located in the

IVR. In this way, the CPU vector points directly to the right interrupt routine without any software overhead.

As the priority decoder is always active, the arbitration is never stopped. It may happen that an interrupt event asserts low the ARM7TDMI nIRQ line and if between the nIRQ line asserted low and the IVR read operation a new highest priority event appears, the IVR will have the value corresponding to the highest priority pending interrupt when the IVR is read.

It is not mandatory to read the IVR and to branch directly to the right interrupt routine with the instruction at address 0x18. An alternative solution could be to branch to a single interrupt entry point and to read the IVR register later on. The only mandatory operations are to first read the IVR once only, then to clear the corresponding pending bit. From an EIC standpoint, the interrupt is acknowledged when the IVR is read and is completed when the corresponding pending bit is cleared. From an ARM7TDMI core stand point the interrupt is acknowledged when the ARM7TDMI nIRQ line is asserted low and is completed when the exception return sequence is executed.

The EIC nIRQ line is equivalent to the ARM7TMDI nIRQ line but in addition it is masked by the EIC global enable bit (IRQ_EN). The EIC nIRQ line can be asserted low but if the global EIC enable bit is not set the ARM7TDMI nIRQ line will not be asserted low.

The IPR is a read/clear register, so writing a ‘0’ has no effect, while writing a ‘1’ resets the related bit. Therefore, refrain from using read-modify instructions to avoid corruption of the IPR status. Most of the EIC pending bits are related to a peripheral pending bit. The peripheral pending bit must be cleared prior to clearing the EIC pending bit. Otherwise the EIC pending bit will be set again and the interrupt routine will be executed twice.

7.1.8 EIC IRQ notes

Reading the IVR while the FSM is in READY state will have no effect. The value read will be unpredictable. Actually, the EIC assigns the IVR value to one of the IRQ routine addresses by default.

The IVR can also be unpredictable while the FSM is in WAIT state. This has to be avoided as the CPU would execute one interrupt routine while the value in the IVR register is not relevant. It would result in an EIC stack corruption as the corresponding pending bit could be reset.

There are several cases where it may happen:

- When the main program raises the main program priority level to a value equal to or higher than the current highest priority pending channel. Interrupts have to be disabled globally during this operation.
- When lowering the priority of a pending channel priority by modifying the SIRn register to a value equal or lower than the main program priority.
- When the software clears some pending bits without taking care to execute the standard interrupt routine sequence.

In such cases the priority decoder loses the winner while the EIC nIRQ line is still being asserted. Only reading the IVR will release the EIC nIRQ line.

The CPU will execute the interrupt routine without having a relevant value in the IVR register, possibly corrupting the stack. If the corresponding pending bit is reset, it will not be possible to execute the EIC stack pop operation.

The normal way to process an interrupt event is to read the IVR register only one time from the interrupt routine, otherwise it may corrupt the EIC stack. Before exiting the interrupt

routine the corresponding peripheral and the EIC pending bits must be cleared. As soon as the IVR is read the application software can read the CICR register to know which interrupt routine is currently executing, as long as the IVR register is not used as a routine pointer.

If the IVR is not read in the interrupt routine the nIRQ line will not be released and the interrupt will be executed twice. If the pending bit is already cleared the EIC stack will be corrupted as it will not be able to perform the pop operation.

Inside a routine it is not an issue to clear pending bits having a lower priority level than the current one, as the nIRQ line is not asserted low in that case. It can be an issue to clear a pending bit having a higher priority level as the EIC nIRQ line is already asserted low, and when interrupts are re-enabled, the EIC stack will be corrupted.

Clearing a pending bit of an interrupt already in the stack will corrupt the stack.

In the main program, if the global interrupts are disabled, all interrupt sources are disabled and all pending bits are cleared, if the nIRQ was already asserted low as soon as the global interrupts are enabled the CPU executes an interrupt routine. In this situation the IVR read will have an unpredictable value, corrupting the EIC stack.

There are two safe ways to clear pending bits without executing the interrupt routine. The first one is to clear them from an IRQ routine having the higher priority level. By this way, the EIC nIRQ line is guaranteed to be released. The second one applies only from the main routine when there is no nested interrupt (EIC stack empty). It consists in following these steps:

1. The global interrupts are disabled
2. Disable all the interrupt channel bits (IER0 = IER1 = 0)
3. IVR is read
4. Read CICR
5. Clear all IPR0 or IPR1 pending bits (IPR0=0 or IPR1=0), it depends on CICR read. The software has to clear first the register not related to CICR read. If the channel read is below 32, IPR1 must be cleared first.
6. Clear the second EIC pending bit register.
7. Enable IER0 and IER1 according to the user application.
8. Enable global interrupts.

If the FSM is in READY state, reading IVR in operation 3) will not do anything. Clearing pending bits will not pop anything as this code is executed from the main routine with no nested interrupts (this is the reason why this method is not recommended from an interrupt routine as clearing all the pending bits will abort the current EIC interrupt routine).

If the FSM is in WAIT state the IVR read will correspond to the highest priority level channel, the main CIPR register will be pushed into the stack, and the main program will have the priority of the highest pending interrupt. The EIC nIRQ is released and will not be asserted again if the right IPR register is cleared first. For example if the IVR read correspond to the channel 10 the IPR1 must be cleared first, otherwise, as soon as IPR0 is cleared, the EIC will pop the previous context and any pending interrupts in IPR1 may assert the nIRQ line.

All EIC pending bits can be cleared including the ones that the user application wants to address later on. The user code needs to make sure that, for those interrupts, the peripheral pending bit is not cleared. By this way, the corresponding EIC pending bits will be set again. As all EIC pending bits are cleared, the EIC stack is guaranteed to pop properly. An alternative solution is to make sure that the EIC pending bit corresponding to the IVR read is cleared.

The operation 3), reading the IVR, is done from the main program and not from an interrupt. It is the only exception where IVR should be read outside an interrupt routine.

7.2 FIQ mechanism

Compare to the EIC IRQ mechanism the EIC FIQ mechanism does not provide any automatic vectoring and software priority level to each FIQ interrupt source. It provides a global FIQ enable bit, an enable and a pending bit per FIQ channel.

There are few differences between the global F bit from the CPSR ARM core register and the global FIQ enable bit from the EIC. The F bit can not be modified in user mode while the EIC global FIQ enable bit is always accessible in any modes. In addition the F bit does not modify the nFIQ internal signal level, it just masks the signal to the core while the EIC global FIQ enable bit act on the nFIQ signal level. The nFIQ signal is always inactive as soon as the global EIC FIQ enable bit is reset, and is active as soon as the global EIC FIQ enable bit is set along with at least one FIQ pending bit.

In order for an FIQ channel source to be pending, the corresponding FIQ enable bit must be set. Clearing the FIQ channel enable bit while the corresponding pending bit is set will not clear the pending bit and the channel will stay active until the pending bit is cleared.

In order for the CPU to vector at the address 0x1C (FIQ exception vector address) the F bit and the global EIC FIQ enable bit must be enabled and at least one FIQ pending bit must be active. Otherwise the CPU will not enter the FIQ exception routine.

7.3 Register programming

Here are a few guideline steps on how to program the EIC's registers in order to get the controller up and running quickly. In the following, it is assumed to deal with standard interrupts and that the user wants, for example, to detect an interrupt on channel #52, in which the priority has been assigned to be 5.

First of all, it is necessary to assign the priority and the jump address data related to the interrupt channel #52. Therefore:

- Set in field SIPL of SIR52 the binary "0101", i.e. priority of 5 (it must be not 0 to have an IRQ to be generated).

Two registers are involved to supply the channel interrupt vector data to the EIC controller (IVR[31:16] and SIR52[31:16]:

- Write in SIR52[31:16], i.e. in the upper part of the SIR register related to channel #52, the memory address offset (or the jump offset) where the Interrupt Service Routine, related to interrupt channel #7, starts.
- Insert the base jump address (or the jump opcode) in the most significant half of the IVR register, i.e. IVR[31:16].

Finally, response to the interrupt must be enabled both at the global level and at the single interrupt channel level. To do so these steps shall be followed:

- Set the IRQ_EN bit of ICR to 1
- Set bit # 20 of IER1 to 1

As far as the FIQ interrupts are concerned, since those interrupts do not have any vectorization nor priority, only the first two steps above are involved. Supposing the user wants to enable FIQ channel #2:

- Set the IRQ_EN bit of ICR to 1
- Set bit6 of FIE in FIR register to 1.

Note: It is recommended to disable IRQ/FIQ interrupts ('I' and 'F' bits of the CPSR ARM7TDMI processor register) before updating the EIC configuration registers. This is to avoid the processor starting to serve new interrupts while you are updating the EIC registers (most operations take two clock cycles). When the EIC has been successfully updated, you can safely reset the ARM CPSR 'I'/'F' bits and re-enable IRQ/FIQ interrupt processing.

7.4 Application note

Every Interrupt Service Routine (ISR) allowing nested interrupts should be composed by the following blocks of code.

- A **Header routine** to enter ISR. It must be:

```
1) STMFD sp!,{r5,lr}The workspace r5 plus the current return address lr_irq is
pushed into the system stack.
2) MRS r5,spsr Save the spsr into r5
3) STMFD sp!,{r5}Save r5
4) MSR cpsr_c,#0x1FRe-enable IRQ, go into system mode
5) STMFD sp!,{lr}Save lr_sys for the system mode
```

Note: r5 is a generic register among those available, namely r0 up to r12. Since there is no way to save SPSR directly into the ARM's stack, the operation is executed in two steps using r5 as transition help register.

- The ISR **Body routines**.
- A **Footer routine** to exit ISR. It must be:

```
1) LDMFD sp!,{lr}Restore lr_sys for the system mode
2) MSR cpsr_c,#0xD2 Disable IRQ, move to IRQ mode
3) Clear pending bit in EIC (using the proper IPRx)
4) LDMFD sp!,{r5}Restore r5
5) MSR spsr,r5 Restore Status register spsr
6) LDMFD sp!,{r5,lr}Restore status lr_irq and workspace
7) SUBS pc,lr,#4 Return from IRQ and interrupt re-enabled
```

In the following two paragraphs some comments on the code lines introduced above and some hints useful when realizing subroutines to be invoked by an ISR are reported.

7.4.1 Avoiding LR_sys and r5 registers content loss

A first example refers to a LR_sys content loss problem: it is assumed that an ISR without instruction 5) in the header routine (and consequently without instruction 1) in the footer routine) has just started; the following happens:

- Instruction 4) is executed (so system mode is entered)
- If before the interrupt event the main program was in a leaf routine (no function call in the routine) meaning that the system link register is not pushed into the stack, as soon as the system mode is entered in the ISR the link register could be corrupted if there a function call happens. It will not be possible to return from the leaf routine in the main program.

The work-around to avoid such a dangerous situation is to insert line 5) at the end of header routine and consequently line 1) at the beginning of footer routine.

Similar reasons could lead register **r5** to be corrupted. To fix this problem, lines 3) in header and 4) in footer should be added.

7.4.2 Hints about subroutines used inside ISRs

A case in which a subroutine is called by an ISR is hereafter considered.

Supposing that such a kind of procedure starts with an instruction like:

```
STMFDSP!, {..., LR}
```

probably it will end with:

```
LDMFDSP!, {..., LR}

MOVPC, LR
```

If a higher priority IRQ occurs between the last two instructions, and the new ISR calls in turn another subroutine, LR content will be lost: so, when the last IRQ ends, the previously interrupted subroutine will not return to the correct address.

To avoid this, the previous two instructions shall be replaced with the unique instruction:

```
LDMFDSP!, {..., PC}
```

which automatically moves stored link register directly into the program counter, making the subroutine correctly return.

7.5 Interrupt latency

As soon as an interrupt request is generated (either from external interrupt source or from an on-chip peripheral), the request itself must go through three different stages before the interrupt handler routine can start. The interrupt latency can be seen as the sum of three different contributions:

- Latency due to the synchronisation of the input stage. This logic can be present (e.g. synchronization stage on external interrupts input lines) or not (e.g. on-chip interrupt request), depending on the interrupt source. Either zero or two clock cycles delay are related to this stage.
- Latency due to the EIC itself. Two clock cycles are related to this stage.
- Latency due to the ARM7TDMI interrupt handling logic (refer to the documentation available on www.arm.com).

Table 20. EIC Interrupt latency (in clock cycles)

	Synch. stage		EIC STAGE
	min	max	
FIQ	0	2	2
IRQ			

7.6 Register description

Reading from the “Reserved” field in any register will return ‘0’ as result. Attempt to write in the same field has no effect.

7.6.1 Interrupt control register (EIC_ICR)

Address Offset: 00h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

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

Bits 31:2 = Reserved, must be kept at reset value (0).

Bit 1= **FIQ_EN:** *Global FIQ output ENable bit*
This bit enables FIQ output request to the CPU.
1: Enhanced Interrupt Controller FIQ output request to CPU is enabled
0: Enhanced Interrupt Controller FIQ output request to CPU is disabled, even if the EIC logic has detected valid and enabled fast interrupt requests at its inputs

Note: It is recommended to set the ARM7TDMI CPSR 'F' bit to disable FIQ interrupts before updating the FIQ_EN bit. This is to avoid the processor starting to serve a new interrupt while the EIC is being disabled (this operation takes two clock cycles). When the EIC has been successfully updated, it is possible to safely reset the ARM CPSR 'F' and re-enable FIQ interrupt processing.

Bit 0 = **IRQ_EN:** *Global IRQ output ENable bit*
This bit enables IRQ output request to the CPU.
1: Enhanced Interrupt Controller IRQ output request to CPU is enabled
0: Enhanced Interrupt Controller IRQ output request to CPU is disabled, even if the EIC logic has detected valid and enabled interrupt requests at its inputs

Note: It is recommended to set the ARM7TDMI CPSR 'I' bit to disable IRQ interrupts before updating the IRQ_EN bit. This is to avoid the processor starting to serve a new interrupt while the EIC is being disabled (this operation takes two clock cycles). When the EIC has been successfully updated, it is possible to safely reset the ARM CPSR 'I' and re-enable IRQ interrupt processing.

The ICR register is a global enable register.

Note: While resetting to ‘0’ FIQ_EN bit simply masks out FIQ output request to CPU, resetting to ‘0’ IRQ_EN bit keeps under reset the FSM which controls IRQ output request to CPU and freezes the FSM that manages the EIC Stack Pointer (Push/Pop control). This is done to make it safe to perform operations such as clearing of unwanted pending IRQs, raising of the priority value in CIPR register, modification of the priority for any IRQ channel by re-writing its related SIR register SIPL field.

7.6.2 Current interrupt channel register (EIC_CICR)

Address Offset: 04h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved										CIC[5:0]					
-										r					

The CICR Register reports the channel ID of the interrupt channel currently serviced. There are 64 possible channel IDs (0 to 63), so the significant register bits are only six (5 down to 0).

After reset, the CICR value is set to ‘0’ and is updated by the EIC logic only after the processor has started servicing a valid IRQ interrupt request e.g. one clock cycle after having read IVR.

To make this happen, some EIC registers must be set as in the following:

- ICR IRQ_EN =1 (to have the nIRQ signal to ARM7TDMI active)
- IER0/IER1 not all 0 (at least one interrupt channel must be enabled)
- Among the interrupt channels enabled by the IERx registers, at least one must have the SIPL field, of the related SIR register, not set to 0, because the EIC generates a processor interrupt request (asserting the nIRQ line) **ONLY IF** it detects an enabled interrupt request whose **priority value is bigger than the CIPR (Current Interrupt Priority Register) value.**

When the nIRQ signal to ARM7TDMI is activated, the CPU will read the EIC IVR (Interrupt Vector Register), and this read operation will advise the EIC logic that the ISR (Interrupt Service Routine) has been initiated and the CICR register can be properly updated.

The CICR value can’t be modified by the CPU (read only register).

Bits 31:6 = Reserved, must be kept at reset value (0).

Bits 5:0 = **CIC[5:0]:** *Current Interrupt Channel*.
Number of the interrupt whose service routine is currently in **execution** phase.

7.6.3 Current interrupt priority register (EIC_CIPR)

Address Offset: 08h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												CIP[3:0]			
-												rw			

The CIPR Register provides the priority value of the main program or the priority value of the interrupt routine currently serviced. There are 16 possible priority values (0 to 15), so the significant register bits are only four (3 down to 0).

After reset, the CIPR value is set to ‘0’ and is updated by the EIC logic only after the processor has started servicing a valid IRQ interrupt request e.g. one clock cycle after having read IVR.

To make this happen, some EIC registers must be set as in the following:

- ICR IRQ_EN =1 (to have the nIRQ signal to ARM7TDMI active)
- IER0/IER1 not all 0 (at least one interrupt channel must be enabled)
- Among the interrupt channels enabled by the IERx registers, at least one must have the SIPL field, of the related SIR register, not set to 0, because the EIC generates a processor interrupt request (asserting the nIRQ line) **ONLY IF** it detects an enabled interrupt request whose **priority value is bigger than the CIPR (Current Interrupt Priority Register) value.**

When the nIRQ signal to ARM7TDMI is activated, the CPU will read the EIC IVR (Interrupt Vector Register), and this read operation will advise the EIC logic that the ISR (Interrupt Service Routine) has been initiated and the CIPR register can be properly updated.

The CIPR value can be modified by the user code, to either rise the priority of the main code (in order to avoid to serve incoming interrupt requests with a priority value lower than a desired one) or to promote a running ISR to a higher level: in this last case, the EIC logic will allow a write to the CIP field of any value higher, or equal, than the priority value associated to the interrupt channel currently serviced.

For example, suppose the nIRQ signal to ARM7TDMI is activated because of an enabled interrupt request on channel #4, whose priority value is 7 (i.e. SIPL of SIR7 is 7); after the processor read of IVR register, the EIC will load the CIP field with 7. Until the interrupt service procedure will be completed, writes of values 7 up to 15 will be allowed, while attempts to modify the CIP content with priority lower than 7 will have no effect.

The user software has to pay attention to avoid a situation where the FSM is in WAIT state and the IVR having an unpredictable value.

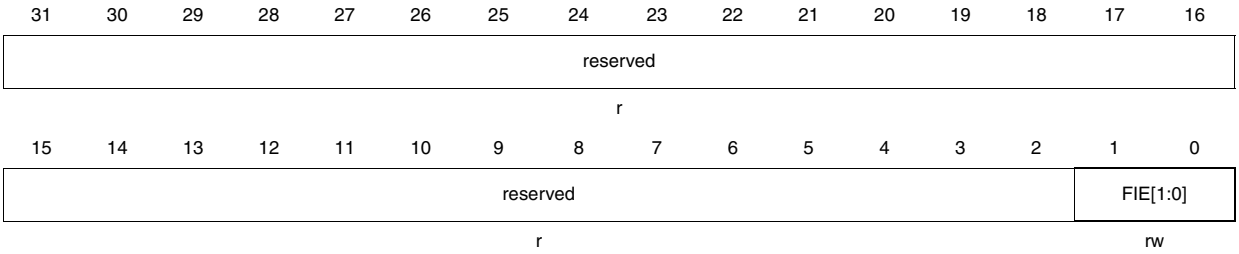
Bits 31:4 = Reserved, must be kept at reset value (0).

Bits 3:0 = **CIP[3:0]:** *Current Interrupt Priority.*
Priority value of the interrupt whose service routine is currently in execution phase.

Note: In order to safely rise current CIPR (both from main code and from an ISR), bit IRQ_EN of ICR register should be cleared first, otherwise the EIC FSM could be led into an unrecoverable state.

7.6.4 Fast interrupt enable register (EIC_FIER)

Address Offset: 10h
Reset value: 0000 0000h



The FIER Register is just an alias of field FIE in FIR register.

Bits 31:2 = Reserved, must be kept at reset value (0).

Bit 1 = **FIE[1]:** *FIQ Channel 1 Interrupt Enable bit*

Bit 0 = **FIE[0]:** *FIQ Channel 0 Interrupt Enable bit*

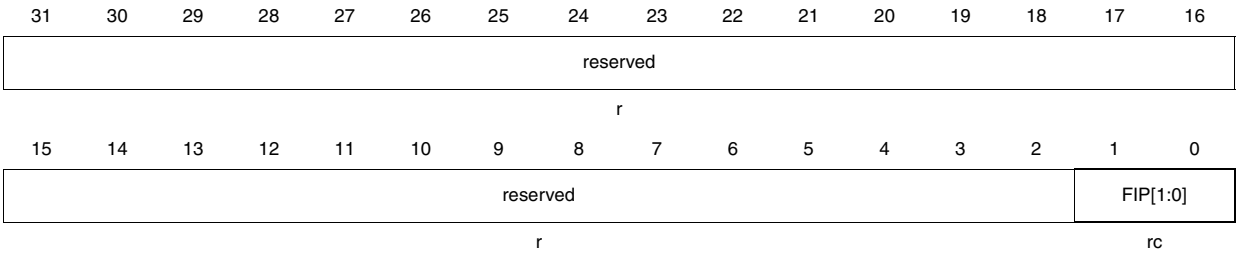
In order to have the controller responding to a request on a specific channel, the corresponding bit in the FIER register must be set to 1.

A '0' value prevents the corresponding pending bit to be set.

1: Fast Interrupt request issued on the specific channel is enabled
0: Fast Interrupt request issued on the specific channel is disabled

7.6.5 Fast interrupt pending register (EIC_FIPR)

Address Offset: 14h
Reset value: 0000 0000h



The FIPR Register is just an alias of field FIP in FIR register.

Bits 31:2 = Reserved, must be kept at reset value (0).

Bit 1 = **FIP[1]:** *Channel 1 Fast Interrupt Pending Bit*

Bit 0 = **FIP[0]:** *Channel 0 Fast Interrupt Pending Bit*

These bits are set by hardware by a Fast interrupt request on the corresponding channel. These bits are cleared only by software, i.e. writing a '0' has no effect, whereas writing a '1' clears the bit (forces it to '0').

1: Fast Interrupt pending
0: No Fast interrupt pending

7.6.6 Interrupt vector register (EIC_IVR)

Address Offset: 18h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IVR[31:16]															
rw															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IVR[15:0]															
r															

The IVR is the EIC register the CPU has to read after detecting the nIRQ signal assertion.

The IVR read operation tells the EIC logic that the interrupt service routine (ISR), related to the pending request, has been initiated.

This means that:

- the nIRQ signal can be de-asserted
- the CIPR and CICR can be updated
- no interrupt requests, whose priority is lower, or equal, than the current one can be processed

Bits 31:16 = **IVR[31:16]**: *Interrupt Vector (High portion).*
Its content does not depend on the interrupt to be serviced, but has to be programmed by the user (see Note) at initialization time and it's common to all the interrupt channels. It is Read/Write.

Bits 15:0 = **IVR[15:0]**: *Interrupt Vector (Low portion).*
Its content depends on the interrupt to be serviced (i.e. the one, in the enabled bunch, with the highest priority), and it's a copy of the SIV (Source Interrupt Vector) value of the SIR (Source Interrupt Register) related to the channel to be serviced. It is Read only.

Note: The EIC logic does not care about the IVR content: from the controller point of view it's a simple concatenation of two 16-bit fields

IVR = IVR[31:16] & SIRn[31:16]

What has to be written in the IVR[31:16] is the higher part of the address pointing to the memory location where the interrupt service routines begin; the single SIRn(31:16) will have to contain the lower 16 bits (offset) of the memory address related to the channel specific ISR.

Reading IVR has acknowledged only when the CPU is not in debug mode and the user code is executing in ARM IRQ mode.

7.6.7 Fast interrupt register (EIC_FIR)

Address Offset: 1Ch
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved											FIP[1:0]		reserved	FIE[1:0]	
-											rc		-	rw	

In order for the controller to react to the 2 fast-interrupt (FIQ) channels the enable bits 1 and 0 must be set to 1. The field 4 and 3 keeps the information on the interrupt request of the specific channel.

Bit 31:5 = Reserved, must be kept at reset value (0).

Bit 4 = **FIP[1]:** *Channel 1 Fast Interrupt Pending Bit*

Bit 3 = **FIP[0]:** *Channel 0 Fast Interrupt Pending Bit*

These bits are set by hardware by a Fast interrupt request on the corresponding channel. These bits are cleared only by software, i.e. writing a ‘0’ has no effect, whereas writing a ‘1’ clears the bit (forces it to ‘0’).
1: Fast Interrupt pending
0: No Fast interrupt pending

Bit 1 = **FIE[1]:** *FIQ Channel 1 Interrupt Enable bit*

Bit 0 = **FIE[0]:** *FIQ Channel 0 Interrupt Enable bit*

In order to have the controller responding to a request on a specific channel, the corresponding bit in the FIR register must be set to 1.

A ‘0’ value prevents the corresponding pending bit to be set.

1: Fast Interrupt request issued on the specific channel is enabled
0: Fast Interrupt request issued on the specific channel is disabled

7.6.8 Interrupt enable register 0 (EIC_IER0)

Address Offset: 20h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IER[31:16]															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IER[15:0]															
rw															

Bit 31 = **IER[31]:** *Channel 31 Interrupt Enable bit*

Bit 30 = **IER[30]:** *Channel 30 Interrupt Enable bit*

...

Bit 1 = **IER[1]:** *Channel 1 Interrupt Enable bit*

Bit 0 = **IER[0]:** *Channel 0 Interrupt Enable bit*

The IER0 is a 32 bit register: it provides an enable bit for each of the lower 32 EIC interrupt input channels.

In order to enable the interrupt response to a specific interrupt input channel the corresponding bit in the IER0 register must be set to '1'.

A '0' value prevents the corresponding pending bit to be set.

1: Input channel enabled
0: Input channel disabled

7.6.9 Interrupt enable register 1 (EIC_IER1)

Address Offset: 24h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IER[63:48]															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IER[47:32]															
rw															

Bit 31= **IER[63]:** *Channel 63 Interrupt Enable bit*

Bit 30 = **IER[62]:** *Channel 62 Interrupt Enable bit*

...

Bit 1 = **IER[33]:** *Channel 33 Interrupt Enable bit*

Bit 0 = **IER[32]:** *Channel 32 Interrupt Enable bit*

The IER1 is a 32 bit register: it provides an enable bit for each of the upper 32 EIC interrupt input channels.

In order to enable the interrupt response to a specific interrupt input channel the corresponding bit in the IER1 register must be set to '1'.

A '0' value prevents the corresponding pending bit to be set.

1: Input channel enabled
0: Input channel disabled

7.6.10 Interrupt pending register 0 (EIC_IPR0)

Address Offset: 40h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IPR[31:16]															
rc															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IPR[15:0]															
rc															

Bit 31= **IPR[31]:** *Channel 31 Interrupt Pending bit*

Bit 30 = **IPR[30]:** *Channel 30 Interrupt Pending bit*

...

Bit 1 = **IPR[1]:** *Channel 1 Interrupt Pending bit*

Bit 0 = **IPR[0]:** *Channel 0 Interrupt Pending bit*

The IPR0 is a 32 bit register: it provides a pending bit for each of the lower 32 EIC interrupt input channels.

This is where the information about the channel interrupt status is kept: if the corresponding bit in the enable register IER0 has been set, the IPR0 bit set high (active) means that the related channel has asserted an interrupt request, that has not been serviced yet.

The bits are Read/Clear, i.e. writing a ‘0’ has no effect, whereas writing a ‘1’ clears the bit (forces it to ‘0’).

1: Interrupt pending
0: No interrupt pending

Note: 1 Before exiting the ISR (Interrupt Service Routine), the software must be sure to have cleared the EIC IPR0 bit related to the executed routine: this bit clear will be read by the EIC logic as End of Interrupt (EOI) sequence, and will allow the interrupt stack pop and new interrupts processing .

2 The Interrupt Pending bits must be handled with care because the EIC State Machine, and its internal priority hardware stack, could be forced to a not recoverable condition if unexpected pending bit clear operations are performed.

Example1

Suppose that one or more interrupt channels are enabled, with a priority higher than zero; as soon as an interrupt request arises, the EIC FSM processes the new input and asserts the nIRQ signal. If, before reading the IVR (0x18) register, for any reason, the CPU clears the pending bits, the nIRQ signal will remain asserted the same, even if no more interrupts are pending.
The only way to reset the nIRQ line logic is to read the IVR (0x18) register, to send a Soft Reset to the EIC or to reset bit IRQ_EN of ICR register.

Example2

Suppose that one or more interrupt channels are enabled, with a priority higher than zero; as soon as an interrupt request arises, the EIC FSM processes the new input and asserts the nIRQ signal. If, after reading the IVR (0x18) register, for any reason, the CPU clears the

pending bit related to the serviced channel, before completing the Interrupt Service Routine, the EIC logic will detect an End Of Interrupt command, will send a pop request to the priority stack and a new interrupt, even of lower priority, will be processed.
To close an interrupt handling section (EOI), the interrupt pending clear operation must be performed, at the end of the related Interrupt Service Routine, on the pending bit related to the serviced channel; on the other hand, as soon as the pending bit of the serviced channel is cleared (even by mistake) by the SW, the EOI sequence is entered by the EIC logic.

3 In order to safely clear a pending bit of an IRQ not currently serviced, bit IRQ_EN of ICR register should be cleared first. If this is not done, the EIC FSMs could be led into an unrecoverable state.
Anyway, while in main program this operation has no drawbacks, when performed inside an IRQ routine it is very important not to clear by mistake the IPRn bit related to the currently served IRQ: since IRQ_EN bit freezes the Stack, the pop operation for the current IRQ won't be done and won't be even possible in future when IRQs will be re-enabled.

7.6.11 Interrupt pending register 1 (EIC_IPR1)

Address Offset: 44h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IPR[63:48]															
rc															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IPR[47:32]															
rc															

Bit 31= **IPR[63]:** *Channel 63 Interrupt Pending bit*

Bit 30 = **IPR[62]:** *Channel 62 Interrupt Pending bit*

...

Bit 1 = **IPR[33]:** *Channel 33 Interrupt Pending bit*

Bit 0 = **IPR[32]:** *Channel 32 Interrupt Pending bit*

The IPR1 is a 32 bit register: it provides a pending bit for each of the upper 32 EIC interrupt input channels.

This is where the information about the channel interrupt status is kept: if the corresponding bit in the enable register IER1 has been set, the IPR1 bit set high (active) means that the related channel has asserted an interrupt request, that has not been serviced yet.

The bits are Read/Clear, i.e. writing a '0' has no effect, whereas writing a '1' clears the bit (forces it to '0').

1: Interrupt pending
0: No interrupt pending

Note: 1 Before exiting the ISR (Interrupt Service Routine), the software must be sure to have cleared the EIC IPR1 bit related to the executed routine: this bit clear will be read by the EIC logic as End of Interrupt (EOI) sequence, and will allow the interrupt stack pop and new interrupts processing .

Note2 and **Note3**: please refer to IPR0 section.

7.6.12 Source interrupt registers - channel n (EIC_SIRn)

Address Offset: 60h to 15Ch
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SIV[31:16]															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												SIPL[3:0]			
-												rw			

There are 64 different SIRn registers, as many as the input interrupt channels are.

Bits 31:16 = **SIV[31:16]:** *Source Interrupt Vector for interrupt channel n (n = 0... 63).*
This field contains the interrupt channel depending part of the interrupt vector, that will be provided to the processor when the Interrupt Vector Register (Address 0x18h) is read.

The SIV will have to be loaded with the lower 16 bits (offset) of the memory address related to the first ISR instruction.

Bits 15:4: Reserved, must be kept at reset value (0).

Bits 3:0 = **SIPL[3:0]:** *Source Interrupt Priority Level for interrupt channel n (n = 0... 63).*
These 4 bits allow to associate the interrupt channel to a priority value between 0 and 15. The reset value is 0.

Note: To be processed by the EIC priority logic, an interrupt channel must have a priority level higher than the current interrupt priority (CIP); the lowest value CIP can have is 0, so all the interrupt sources that have a priority level equal to 0 will never generate an IRQ request, even if properly enabled.

7.7 EIC register map

Table 21. EIC register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ICR	reserved																														FIQ_EN	IRQ_EN
4	CICR	reserved																												CIC[3:0]			
8	CIPR	reserved																												CIP[3:0]			
C		reserved																															
10	FIER	reserved																													FIE[2:0]		
14	FIPR	reserved																													FIP[2:0]		
18	IVR	Jump Instruction Opcode or Jump Base Address																Jump Offset															
1C	FIR	reserved																										FIP[2:0]			FIE[2:0]		
20	IER0	IER[31:0]																															
24	IER1	IER[63:32]																															
40	IPR0	IPR[31:0]																															
44	IPR1	IPR[63:32]																															
60	SIR0	SIV0[31:16]																reserved													SIPL0[3:0]			
64	SIR1	SIV1[31:16]																reserved													SIPL1[3:0]			
68	SIR2	SIV2[31:16]																reserved													SIPL2[3:0]			
6C	SIR3	SIV3[31:16]																reserved													SIPL3[3:0]			
70	SIR4	SIV4[31:16]																reserved													SIPL4[3:0]			
74	SIR5	SIV5[31:16]																reserved													SIPL5[3:0]			
78	SIR6	SIV6[31:16]																reserved													SIPL6[3:0]			
7C	SIR7	SIV7[31:16]																reserved													SIPL7[3:0]			
80	SIR8	SIV8[31:16]																reserved													SIPL8[3:0]			
84	SIR9	SIV9[31:16]																reserved													SIPL9[3:0]			
88	SIR10	SIV10[31:16]																reserved													SIPL10[3:0]			
8C	SIR11	SIV11[31:16]																reserved													SIPL11[3:0]			
90	SIR12	SIV12[31:16]																reserved													SIPL12[3:0]			
94	SIR13	SIV13[31:16]																reserved													SIPL13[3:0]			
98	SIR14	SIV14[31:16]																reserved													SIPL14[3:0]			
9C	SIR15	SIV15[31:16]																reserved													SIPL15[3:0]			
A0	SIR16	SIV16[31:16]																reserved													SIPL16[3:0]			
A4	SIR17	SIV17[31:16]																reserved													SIPL17[3:0]			
A8	SIR18	SIV18[31:16]																reserved													SIPL18[3:0]			
AC	SIR19	SIV19[31:16]																reserved													SIPL19[3:0]			
B0	SIR20	SIV20[31:16]																reserved													SIPL20[3:0]			
B4	SIR21	SIV21[31:16]																reserved													SIPL21[3:0]			
B8	SIR22	SIV22[31:16]																reserved													SIPL22[3:0]			

Table 21. EIC register map (continued)

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BC	SIR23	SIV23[31:16]																reserved												SIPL23[3:0]			
C0	SIR24	SIV24[31:16]																reserved												SIPL24[3:0]			
C4	SIR25	SIV25[31:16]																reserved												SIPL25[3:0]			
C8	SIR26	SIV26[31:16]																reserved												SIPL26[3:0]			
CC	SIR27	SIV27[31:16]																reserved												SIPL27[3:0]			
D0	SIR28	SIV28[31:16]																reserved												SIPL28[3:0]			
D4	SIR29	SIV29[31:16]																reserved												SIPL29[3:0]			
D8	SIR30	SIV30[31:16]																reserved												SIPL30[3:0]			
DC	SIR31	SIV31[31:16]																reserved												SIPL31[3:0]			
E0	SIR32	SIV32[31:16]																reserved												SIPL32[3:0]			
E4	SIR33	SIV33[31:16]																reserved												SIPL33[3:0]			
E8	SIR34	SIV34[31:16]																reserved												SIPL34[3:0]			
EC	SIR35	SIV35[31:16]																reserved												SIPL35[3:0]			
F0	SIR36	SIV36[31:16]																reserved												SIPL36[3:0]			
F4	SIR37	SIV37[31:16]																reserved												SIPL37[3:0]			
F8	SIR38	SIV38[31:16]																reserved												SIPL38[3:0]			
FC	SIR39	SIV39[31:16]																reserved												SIPL39[3:0]			
400	SIR40	SIV40[31:16]																reserved												SIPL40[3:0]			
104	SIR41	SIV41[31:16]																reserved												SIPL41[3:0]			
108	SIR42	SIV42[31:16]																reserved												SIPL42[3:0]			
10C	SIR43	SIV43[31:16]																reserved												SIPL43[3:0]			
110	SIR44	SIV44[31:16]																reserved												SIPL44[3:0]			
114	SIR45	SIV45[31:16]																reserved												SIPL45[3:0]			
118	SIR46	SIV46[31:16]																reserved												SIPL46[3:0]			
11C	SIR47	SIV47[31:16]																reserved												SIPL47[3:0]			
120	SIR48	SIV48[31:16]																reserved												SIPL48[3:0]			
124	SIR49	SIV49[31:16]																reserved												SIPL49[3:0]			
128	SIR50	SIV50[31:16]																reserved												SIPL50[3:0]			
12C	SIR51	SIV51[31:16]																reserved												SIPL51[3:0]			
130	SIR52	SIV52[31:16]																reserved												SIPL52[3:0]			
134	SIR53	SIV53[31:16]																reserved												SIPL53[3:0]			
138	SIR54	SIV54[31:16]																reserved												SIPL54[3:0]			
13C	SIR55	SIV55[31:16]																reserved												SIPL55[3:0]			
140	SIR56	SIV56[31:16]																reserved												SIPL56[3:0]			
144	SIR57	SIV57[31:16]																reserved												SIPL57[3:0]			
148	SIR58	SIV58[31:16]																reserved												SIPL58[3:0]			
14C	SIR59	SIV59[31:16]																reserved												SIPL59[3:0]			
150	SIR60	SIV60[31:16]																reserved												SIPL60[3:0]			

Table 21. EIC register map (continued)

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
154	SIR61	SIV61[31:16]																reserved												SIPL61[3:0]			
158	SIR62	SIV62[31:16]																reserved												SIPL62[3:0]			
15C	SIR63	SIV63[31:16]																reserved												SIPL63[3:0]			

See *Table 2 on page 17* for base address

7.8 External interrupt pins INT[15:0]

The STR73x device provides sixteen external interrupt pins [INT15:0]. You can configure them individually as edge-triggered or level-triggered.

7.8.1 Edge-triggered external interrupts

In case of interrupt “edge“ triggered, you can select the interrupt trigger event to be rising edge, falling edge or both. This event will generate a pulse (one system clock cycle long) that will set the corresponding EIC interrupt pending bit (if the related mask bit is enabled). After the reset of the interrupt pending bit (performed by the user interrupt sub routine), a new interrupt will be issued to EIC only if a new edge event occurs on the external interrupt line.

7.8.2 Level-triggered external interrupts

In case of interrupt “level“ triggered, the user may select to issue an interrupt request to EIC module when either a “high” level or a “low” level is applied on external interrupt line. The selected active level will set the correspondent EIC interrupt pending bit (if the related mask bit is enabled). After the reset of the interrupt pending bit (performed by user ISR), a new interrupt will be issued to EIC if the external interrupt line is kept at the selected active level.

Each external interrupt source has three configuration registers: CFG_EITE2, CFG_EITE1 and CFG_EITE0 mapped in the Configuration register block. Refer to *Section 4.2: External interrupt request configuration registers on page 55*.

The selection of the desired event on INTn (n=0,15) triggering interrupt request on EIC module is achieved programming the generic bit n of three configuration registers, as summarized in *Table 22*.

Table 22. External interrupt configuration table

Control bit	Configuration							
EITE2(n)	1	1	1	1	0	0	0	0
EITE1(n)	1	1	0	0	1	1	0	0
EITE0(n)	1	0	1	0	1	0	1	0
Event on INTn input channel triggering Interrupt Request on EIC	INTn channel disabled	INTn rising and falling edges	INTn rising edge	INTn falling edge	INTn HIGH level	INTn LOW level	INTn HIGH level (1)	INTn LOW level

1. Reset Configuration

7.9 Wake-up interrupt unit (WIU)

The main function of the Wake-Up Interrupt Unit (WIU) is to manage the external wake-up lines. The WIU is connected to the IRQ40 channel of the EIC module.

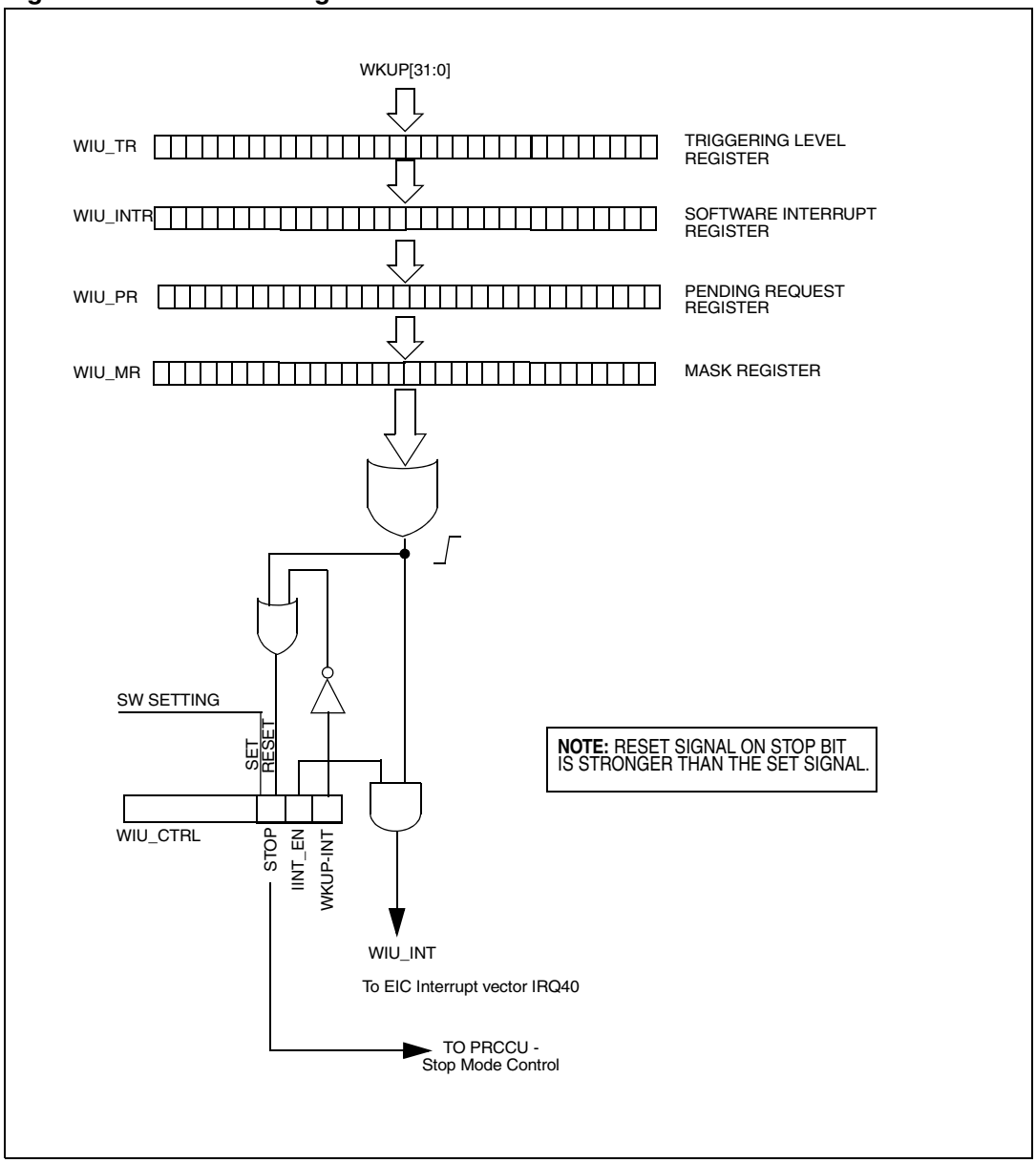
Using the WIU registers, 32 I/O ports can be programmed as external interrupt lines or as wake-up lines, able to wake-up the MCU from WFI or Stop mode.

Some external interrupt lines are mapped to I/O ports that can be enabled as inputs to the CAN, I^2C, BSPI or UART peripherals. This means you can program it so that any activity on these serial buses will wake-up the MCU from WFI or Stop mode.

7.9.1 Features

- External Wake-up lines can be used to wake-up the system from Stop mode
- Programmable selection of Wake-up or Interrupt
- Programmable Wake-up trigger edge polarity
- All Wake-up Lines individually maskable
- Wake-up interrupt generated by software

Figure 25. WIU block diagram



7.9.2 Functional description

7.9.3 Interrupt mode

To configure the 32 lines as interrupt sources, use the following procedure:

1. Configure the mask bits of the 32 wake-up lines (WIU_MR).
2. Configure the triggering edge of the wake-up lines (WIU_TR).
3. In the EIC registers, enable the IRQ40 interrupt channel so an interrupt coming from one of the 32 wake-up lines can be correctly acknowledged.
4. Clear the WKUP-INT bit in the WIU_CTRL register to disable Wake-up Mode.
5. Set the INT_EN bit in the WIU_CTRL register to enable the 16 wake-up lines as external interrupt source lines.

7.9.4 Wake-up mode selection

To configure the 32 lines as wake-up sources, use the following procedure:

1. Configure the mask bits of the 32 wake-up lines (WIU_MR).
2. Configure the triggering edge registers of the wake-up lines (WIU_TR).
3. If an interrupt routine is to be executed after a wake-up event, then enable the IRQ40 interrupt channel using the EIC registers. Otherwise, if the wake-up event only restarts executing of the code from where it was stopped, the IRQ40 interrupt channel must be masked.
4. Since the PRCCU can generate an interrupt request when exiting from Stop mode, take care to mask it if the wake-up event is only to restart code execution.
5. Set the WKUP-INT bit in the WIU_CTRL register to select Wake-up Mode.
6. Set the INT_EN bit in the WIU_CTRL register to enable the 32 wake-up lines as external interrupt source lines.
7. Write the sequence 1,0,1 to the STOP bit of the WIU_CTRL register. This is the STOP bit setting sequence. Pay attention that the three write operations are effective even though not executed in a strict sequence (intermediate instructions are allowed): to reset the sequence it is sufficient to write twice a logic ‘0’ to the STOP bit of WIU_CTRL register (corresponding anyway to a bad sequence).

To detect if Stop Mode was entered or not, immediately after the end of the STOP bit setting sequence, poll the PRCCU STOP_I flag bit and the STOP bit itself (WIU_CTRL register).

7.9.5 Stop mode entry conditions

Assuming the device is in Run mode: during the STOP bit setting sequence the following cases may occur:

Case 1: Wrong STOP bit setting sequence

This can happen if an Interrupt request is acknowledged during the STOP bit setting sequence. In this case polling the STOP and STOP_I bits will give:

STOP = 0, STOP_I = 0

This means that the device did not enter Stop mode due to a bad STOP bit setting sequence: the user must retry the sequence.

Case 2: Correct STOP bit setting sequence

In this case the device enters Stop mode.

To exit Stop mode, a wake-up interrupt must be acknowledged. This implies:

STOP = 0, STOP_I = 1

This means that the device entered and exited Stop mode due to an external wake-up line event.

Case 3: A wake-up event on the external wake-up lines occurs during the STOP bit setting sequence

There are two possible cases:

1. Interrupt requests to the CPU are disabled: in this case the device will not enter Stop mode, no interrupt service routine will be executed and the program execution continues from the instruction following the STOP bit setting sequence. The status of STOP and STOP_I bits will be again:

   STOP = 0, STOP_I = 0

   The application can determine why the device did not enter Stop mode by polling the pending bits of the external lines (at least one must be at 1).
2. Interrupt requests to CPU are enabled: in this case the device will not enter Stop mode and the interrupt service routine will be executed. The status of STOP and STOP_I bits will be again:

   STOP = 0, STOP_I = 0

   The interrupt service routine can determine why the device did not enter Stop mode by polling the pending bits of the external lines (at least one must be at 1).

If the device really exits from Stop Mode, the PRCCU STOP_I bit is still set and must be reset by software. Otherwise, if an Interrupt request was acknowledged during the STOP bit setting sequence, the PRCCU STOP_I bit is reset. This means that the system has filtered the Stop Mode entry request.

The WKUP-INT bit can be used by an interrupt routine to detect and to distinguish events coming from Interrupt Mode or from Wake-up Mode, allowing the code to execute different procedures.

To exit Stop mode, it is sufficient that one of the 32 wake-up lines (not masked) generates an event: the clock restarts after the delay needed for the oscillator to restart.

Note: After waking-up from Stop Mode, software can successfully reset the pending bits (edge sensitive), even though the corresponding wake-up line is still active (high or low, depending on the Trigger Event register programming); the user must poll the external pin status to detect and distinguish a short event from a long one (for example keyboard input with keystrokes of varying length).

7.9.6 Programming considerations

The following paragraphs give some guidelines for designing an application program.

7.9.7 Procedure for entering/exiting Stop mode

1. Program the polarity of the trigger event of external wake-up lines by writing register WIU_TR.
2. Check that at least one mask bit (register WIU_MR) is equal to 1 (so at least one external wake-up line is not masked).
3. Reset at least the unmasked pending bits: if unmasked pending bits are not cleared STOP Mode cannot be entered.
4. Set the INT_EN and the WKUP-INT bits in the WIU_CTRL register.
5. To generate an interrupt on the associated channel (IRQ40), set the related enable, mask and priority bits in the EIC registers.
6. Reset the STOP bit in register WIU_CTRL and STOP_I bit in the PRCCU_CFR register.
7. To enter Stop mode, write the sequence 1, 0, 1 to the Stop bit in the WIU_CTRL register. As already said, the three write operations are effective even though not executed in a strict sequence (intermediate instructions are allowed): to reset the sequence it is sufficient to write twice a logic ‘0’ to the STOP bit of WIU_CTRL register (corresponding anyway to a bad sequence).
8. The code to be executed just after the STOP sequence must check the status of the STOP and PRCCU STOP_I bits to determine if the device entered Stop mode or not. If the device did not enter in Stop mode it is necessary to re-loop the procedure from the beginning, otherwise the procedure continues from next point.
9. Poll the wake-up pending bits to determine which wake-up line caused the exit from Stop mode.
10. Clear the wake-up pending bit that was set.

7.9.8 Simultaneous setting of pending bits

It is possible that several simultaneous wake-up events set different pending bits. In order to accept subsequent events on external wake-up/interrupt lines, once the first interrupt routine has been executed, it is necessary to clear at least one pending bit (the corresponding pending bit in the WIU_PR register): this operation allows a rising edge to be generated on the internal line (if there is at least one more pending bit set and not masked) and so to set the interrupt controller pending bit again. A further interrupt on the same channel of the interrupt controller will be serviced depending on the status of the mask bit. Two possible situations may arise:

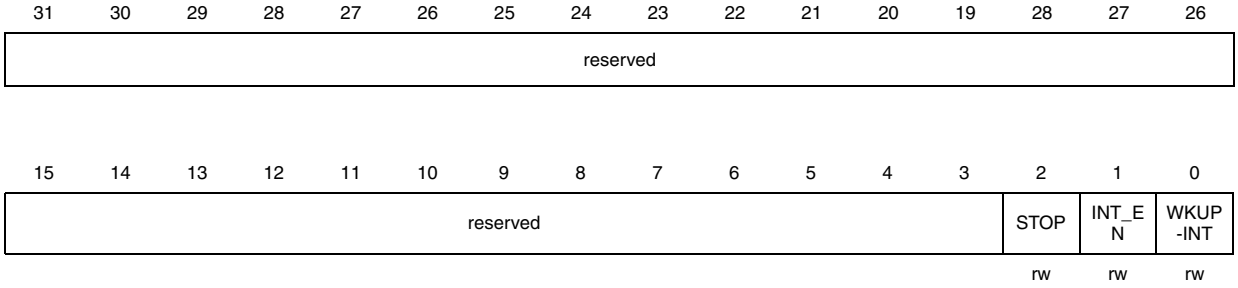
1. The user chooses to reset all pending bits: no further interrupt requests will be generated on the channel. In this case the user has to:
   - Reset the interrupt controller mask bit (to avoid generating a spurious interrupt request during the next reset operation)
   - Reset the WIU_PR register
2. The user chooses to keep at least one pending bit active: at least one additional interrupt request will be generated on the interrupt controller channel. In this case the user has to reset the desired pending bits. This operation will generate a rising edge on the interrupt controller channel and the corresponding pending bit will be set again. An interrupt on this channel will be serviced depending on the status of corresponding mask bit.

7.9.9 Register description

7.9.10 Wake-up control register (WIU_CTRL)

Address Offset: 00h

Reset value: 0000 0000h



Bits 31:3 = Reserved, must be kept at reset value.

Bit 2 = **STOP:** *Stop bit.*
To enter Stop Mode, write the sequence 1,0,1 to this bit with three write operations**.** When a correct sequence is recognized, the STOP bit is set and the PRCCU puts the device in Stop Mode. If the setting sequence is not recognized within 64 clock periods, a timeout counter expires and resets the sequence state machine. In this case the device does not enter Stop Mode and the STOP bit is reset by hardware. The software sequence succeeds only if the following conditions are true:

- The WKUP-INT bit is 1
- All unmasked pending bits are reset
- At least one mask bit is equal to 1 (at least one external wake-up line is not masked).

Otherwise the device cannot enter Stop mode, the program continues executing and the STOP bit remains cleared.
The bit is reset by hardware if, during Stop mode, a wake-up interrupt comes from any of the unmasked wake-up lines. Otherwise the STOP bit is at 1 in the following cases (*Wake-up mode selection on page 105* for details):

- After the first write instruction of the sequence (a 1 is written to the STOP bit)
- At the end of a successful sequence (i.e. after the third write instruction of the sequence)

Caution: If Interrupt requests are acknowledged during the sequence, the system will not enter Stop mode (since the sequence is not completed): at the end of the interrupt service routine, it is recommended to reset the sequence state machine by twice writing a logic ‘0’ to the STOP bit in the WUCTRL register (corresponding anyway to a bad sequence). Otherwise the incomplete sequence will wait for the completion and only the TIMEOUT counter will reset the state machine. You must re-enter the sequence to set the STOP bit.

Caution: Whenever a STOP request is issued to the system, several clock cycles are needed to enter Stop mode (see PRCCU chapter for further details). Hence the execution of the instruction following the STOP bit setting sequence might start before entering Stop mode (consider the ARM7 three-stage pipeline as well). In order to avoid executing any valid instructions after a correct STOP bit setting sequence and before entering the Stop mode, it is mandatory to execute a dummy set of several instructions after the STOP bit setting sequence.

Bit 1 = **INT_EN:** *Global WIU Interrupt Enable.*
This bit is set and cleared by software.
0: WIU interrupts disabled
1: The 32 external wake-up lines enabled as interrupt sources

Caution: To avoid spurious interrupt requests on the IRQ40 channel of the EIC, due to change of interrupt source, it is recommended to clear the corresponding enable bit in the EIC IER register before modifying the INT_EN bit.

Bit 0 = **WKUP-INT:** *Global Wake-up Event Enable.*
This bit is set and cleared by software.
0: Wake-up Event disabled
1: Wake-up from Stop mode enabled

7.9.11 Wake-up mask register (WIU_MR)

Address Offset: 04h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WUM 31	WUM 30	WUM 29	WUM 28	WUM 27	WUM 26	WUM 25	WUM 24	WUM 23	WUM 22	WUM 21	WUM 20	WUM 19	WUM 18	WUM 17	WUM 16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WUM 15	WUM 14	WUM 13	WUM 12	WUM 11	WUM 10	WUM9	WUM8	WUM7	WUM6	WUM5	WUM4	WUM3	WUM2	WUM1	WUM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 = **WUM[31:0]**: *Wake-Up Mask bits*
If WUMx is set, an interrupt and/or a wake-up event (depending on INT_EN and WKUP-INT bits) are generated if the corresponding WUPx pending bit is set. More precisely, if WUMx=1 and WUPx=1 then:

- If INT_EN=1 and WKUP-INT=1 then an interrupt and a wake-up event are generated.
- If INT_EN=1 and WKUP-INT=0 only an interrupt is generated.
- If INT_EN=0 and WKUP-INT=1 only a wake-up event is generated.
- If INT_EN=0 and WKUP-INT=0 neither interrupts nor wake-up events are generated.

If WUMx is reset, no wake-up events can be generated.

7.9.12 Wake-up trigger register (WIU_TR)

Address Offset: 08h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WUT 31	WUT 30	WUT 29	WUT 28	WUT 27	WUT 26	WUT 25	WUT 24	WUT 23	WUT 22	WUT 21	WUT 20	WUT 19	WUT 18	WUT 17	WUT 16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WUT 15	WUT 14	WUT 13	WUT 12	WUT 11	WUT 10	WUT9	WUT8	WUT7	WUT6	WUT5	WUT4	WUT3	WUT2	WUT1	WUT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 = **WUT[31:0]**: *Wake-Up Trigger Polarity bits*
The WUTx bits can be set and cleared by software
0: The corresponding WUPx pending bit will be set upon the falling edge of the input wake-up line.
1: The corresponding WUPx pending bit will be set upon the rising edge of the input wake-up line.

Caution: As the external wake-up lines are edge triggered, no glitches must be generated on these lines.

Caution: If either a rising or a falling edge on an external wake-up line occurs when writing to the WIU_TR register, the pending bit will not be set.

7.9.13 Wake-up software interrupt register (WIU_INTR)

Address Offset: 10h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WU INT31	WU INT30	WU INT29	WU INT28	WU INT27	WU INT26	WU INT25	WU INT24	WU INT23	WU INT22	WU INT21	WU INT20	WU INT19	WU INT18	WU INT17	WU INT16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WU INT15	WU INT14	WU INT13	WU INT12	WU INT11	WU INT10	WU INT9	WU INT8	WU INT7	WU INT6	WU INT5	WU INT4	WU INT3	WU INT2	WU INT1	WU INT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 = **WUINT[31:0]**: *WIU Software Interrupt bits*
WUINTx bits are set by software to implement a software interrupt. Setting one of these bits corresponds settting the same bit in the Pending register (WIU_PR).
WUINTx bits are reset clearing the pending bits, writing a ‘1’ to the WUPx bit.

7.9.14 Wake-up pending register (WIU_PR)

Address Offset: 0Ch

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WUP 31	WUP 30	WUP 29	WUP 28	WUP 27	WUP 26	WUP 25	WUP 24	WUP 23	WUP 22	WUP 21	WUP 20	WUP 19	WUP 18	WUP 17	WUP 16
rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WUP 15	WUP 14	WUP 13	WUP 12	WUP 11	WUP 10	WUP9	WUP8	WUP7	WUP6	WUP5	WUP4	WUP3	WUP2	WUP1	WUP0
rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc

Bits 31:0 = **WUP[31:0]**: *Wake-Up Pending bits*
The WUPx bits are Read/Clear, they are set by hardware on occurrence of the trigger event. They can be reset by software writing a '1'; writing a '0' is ignored.
0: No Wake-Up trigger event occurred
1: Wake-Up Trigger event occurred
WUPx bits may be set also by software, setting the corresponding bits in the Software Interrupt register (WIU_INTR) and choosing the trigger level high (WUTx set to '1').

7.9.15 WIU register map

Table 23. WIU register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	WIU_CTRL	Reserved																													STOP	INT_EN	WKUP-INT
4	WIU_MR	WUM[31:0]																															
8	WIU_TR	WUT[31:0]																															
C	WIU_PR	WUP[31:0]																															
10	WIU_INTR	WUINT[31:0]																															

See *Table 2 on page 17* for base address

8 DMA Controller (DMA)

8.1 Introduction

Four DMA controllers are available in the STR73x. Each DMA controller provides access to 4 data streams. Since peripherals are memory mapped, data transfers from/to peripherals are managed like memory/memory data transfers.

Note: To enable DMA arbitration and transfers, bits 1 and 0 of the Native Arbiter Priority Register have to be set to 1. Refer to Section 9.1 on page 129

Each data stream is associated with a particular peripheral (see *Table 24*). The peripheral, triggers a DMA request, starting the data transfer between the corresponding buffers defined in the stream descriptor. Data stream 3 can alternatively be used as a memory/memory data transfer triggered by a software DMA request, independently from any peripheral activity.

8.2 DMA controller priority

The priority between the different DMA controllers is done through a round-robin strategy where all the DMA requesting the bus have the same priority: if DMA *n* is the last which have accessed the bus, the arbiter will check first if DMA *(n+1) mod 4* requests the bus and if requested grant it. If not requested, it will check DMA *(n+2) mod 4*, then *DMA (n+3) mod 4* and eventually *DMA (n+4) mod 4 = n.*

The priority between the different DMA triggering sources is defined by hardware, source 0 being the highest priority request and source 3 being the lowest one.

Note: A typical "single data" DMA transfer will include the following phases:
- Peripherals to DMA interrupt triggering (~2 cycles)
- DMA bus request, arbitration and memory/peripheral to DMA transfer (~5 cycles).
- Internal DMA latency (1 cycle)
- DMA bus request, arbitration and DMA to memory/peripheral transfer (~5 cycles)
Note that in the case of burst DMA transfer, arbitration and internal DMA latency are seen only once, each supplementary data transfer needing 1 cycle.
Example: burst transfer of 16 bytes organized in 4 words in memory and directed to a 16-bit peripheral (8 half-words).
Transfer_cycles = 2+(5+3)+1+(5+7) = 23 cycles (16 bytes)

8.3 DMA request mapping

Table 24. DMA request mapping

DMA	Stream	DMA triggering source
DMA 0	0	BSPI2RX
	1	BSPI2TX
	2	- / BSPI0RX
	3	- / BSPI0TX

Table 24. DMA request mapping (continued)

DMA	Stream	DMA triggering source
DMA 1	0	TIM0
	1	TIM1
	2	TIM2
	3	TIM3
DMA 2	0	TIM5
	1	TIM6
	2	TIM7
	3	TIM8 / BPI1RX
DMA 3	0	ADC
	1	TIM4
	2	TIM9 / BSPI1TX
	3	memory - to - memory

Bit 4 (DMABSPI0) in the CFG_R0 register has to be set to select the BSPI0 as triggering source for both DMA0 Stream2 and DMA0 Stream3.

When the bit 5 (DMABSPI1) in the CFG_R0 register is set, the BSPI1 is the triggering source for both DMA2 Stream3 and DMA3 Stream2, otherwise TIM8 is the triggering source for DMA2 Stream3 and TIM9 is the triggering source for DMA3 Stream2.

The DMA controller treats bytes in memory as being in Little Endian format: the lowest numbered byte in a word is considered as the least significant byte and the highest numbered byte is the most significant.

8.4 Functional description

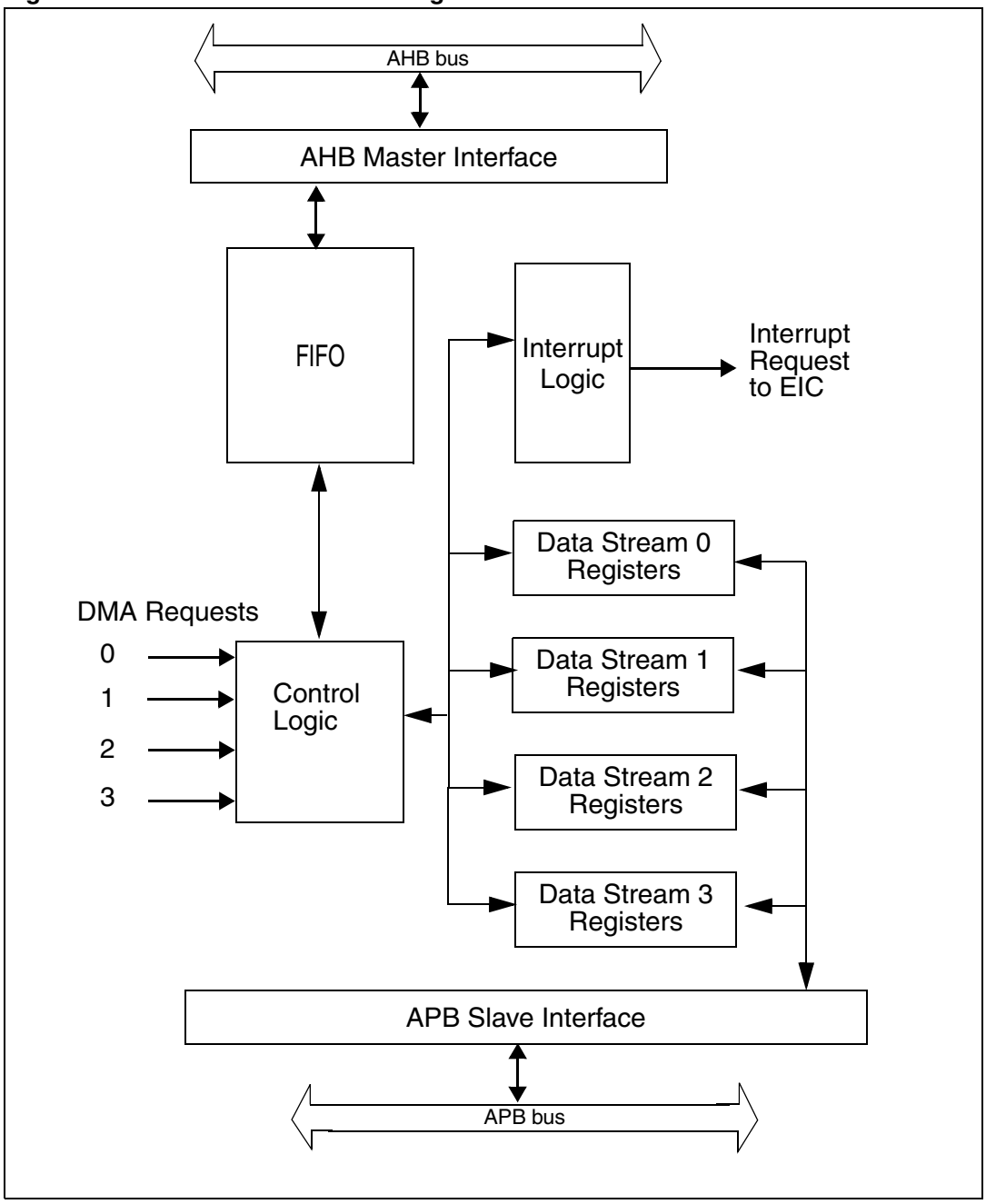
Each DMA Controller is a single channel DMA Controller capable of servicing up to 4 data streams. It has a a single FIFO which is shared by the 4 data streams via the use of priority selection logic as shown in *Table 25*.

Table 25. DMA request priority

DMA Request	Priority
External0	1 (Highest)
External1	2
External2	3
External3 - Internal0/1	4 (Lowest)

DMA Request3 is multiplexed with 2 internal request lines which are selected when a memory to memory transfer is required. The DMA block diagram is shown in *Figure 26*

Figure 26. DMA Controller block diagram



The DMA Controller can be used for a memory to memory transfer, a peripheral to memory transfer or a memory to peripheral transfer.

A DMA data transfer consists of a sequence of DMA data burst transfers. There are two types of burst transfers, the first one is from the source address to the DMA Controller and the second one is from the DMA Controller to the destination address. Each data burst transfer is characterized by the burst length (1, 4, 8, or 16 words) and by the word width (1 byte, 2 bytes or 4 bytes). The DMA data transfer is complete when the programmed total number of bytes has been transferred from the source address to the destination address (Terminal Count register going to zero).

The Control Logic is used to arbitrate between the data streams. It selects the request from the highest priority active data stream, passing the data from the chosen stream to and from the FIFO as required. Each stream has a set of data stream registers which are used by the Control Logic to determine source and destination addresses and the amount and format of data to be transferred. They also provide various other control and status information.

The DMA data stream registers are all 16-bit wide and are accessed via the APB Bus. The control logic can read all of these registers and can write some of them.

There is a single dedicated interrupt line from each DMA Controller to the EIC. It is driven by 4 internal interrupt flags (one per data stream) which are ORed together. They work as follows:

Once a transfer for a specific stream is complete (this condition being detected by its Terminal Count register going to zero) the interrupt flag for the data stream is set. The interrupt logic then generates an interrupt to the EIC telling it that a request for one of the data streams has been completed. At the same time this data stream is disabled (i.e. the DMA Controller clears the enable bit of its control register). The status register must be read to determine which data stream caused the interrupt to be raised. The DMA Controller can now safely reconfigure this data stream (if required) for future transfers. A data stream can be re-enabled via a write to the enable bit of the corresponding Control Register.

The DMA FIFO block consists of a 16 32-bit words deep FIFO plus Data Pack and Data Unpack units. The purpose of the FIFO block is to accommodate bus latency and burst length, and to perform any packing or unpacking operations on the data that may be necessary to accommodate different data-out/data-in width ratios.

DMA Request 3 is multiplexed with 2 internal request signals which can be used for a memory to memory data transfer. To allow their use, the ‘Mem2Mem’ bit in the control register must be set. This tells the DMA Controller that data stream 3 is now configured for an internal, rather than an external, transfer request. An internal data transfer consists of two phases: phase 0 (where internal request 0 is asserted) is used for the memory to FIFO data transfer and phase 1 (where internal request 1 is asserted) is used for the FIFO to memory data transfer. An internal transfer will begin once the ‘Mem2Mem’ bit in the control register is set and data stream 3 is enabled, provided there are no pending requests for any of the other data streams, which all have higher priority. If there are pending requests, then these will be serviced first. When no other requests are pending, phase 0 will start as soon as the channel FIFO can accept the next data burst from the source address. The data packet size is defined in the data stream configuration registers. Phase 1 is asserted if the channel FIFO contains enough data for a next data burst to be sent to the destination address.

Phase 1 will be also started in order to flush the FIFO contents if any of the following conditions occur:

– If the terminal count reaches zero this indicates that the FIFO has received the total number of bytes to be transferred to the destination address. Since this number may be not a integer multiple of the selected burst size, bytes remaining in the FIFO after terminal count reaches zero must be flushed to the destination address.

If stream 3 is disabled (via a write to the enable bit of the control register for this data stream).

In circular buffer mode, the DMA controller reloads the start address when the word counter (Terminal Count register) reaches the end of count and continues the transfer until application software sets the LAST bit, writing in the DMAn_LUBuff register the buffer

location where the last data to be transferred is located. The stream configured in circular mode is controlled by a CIRCULAR flag in the DMAn_Ctrl register (‘0’=normal mode, ‘1’=circular mode), a LAST flag in the DMAn_Last register (‘0’=infinite mode, ‘1’=last buffer sweep), and a DMAn_LUBuff register (read and write).

When a circular buffer is the source of the transfer, the application software do the following:

1. Set the DMA stream configuration registers writing CIRCULAR bit to’1’ and LAST bit to ‘0’.
2. Start feeding the circular buffer using an index to keep a trace of the last buffer location used.
3. When the end of transfer condition occurs, write DMAn_LUBuff with the value of the index and set the LAST bit to ‘1’.
4. The DMA interrupt line will be activated as soon as the DMAn_LUBuff location has been correctly transferred.

When a circular buffer is the destination of the transfer, the application sofware should proceed in a similar way:

1. Set the DMA stream configuration registers writing CIRCULAR bit to’1’ and LAST bit to ‘0’.
2. Start fetching the circular buffer using an index to keep a trace of the last buffer location used.
3. When the end of transfer condition occurs, stop DMA operation writing DMA_EN to ‘0’.
4. The last buffer location to be used will be indicated by DMAn_DeCurr register pair.

Due to the fact that FIFO is always flushed when the end of buffer is reached, it is not possible to use circular buffer mode when the buffer size is not a multiple of the configured burst size. Circular buffer mode cannot be used in the following situations:

- When buffer size is not a multiple of the configured burst size. This is due to the fact that FIFO is always flushed when the end of buffer is reached, and the resulting burst would not be followed by a transfer moving the remaining locations, located at the beginning of circular buffer, to complete the programmed burst size.
- When memory to memory data transfer is configured on stream 3.

8.5 Register description

The DMA registers are accessed via the APB bus and the register data path is 16 bits wide.

Source Base Address Low (DMAn_SOURCELx) (x=0,...,3)

Address Offset: 00h - 40h - 80h - C0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASOURCELx[15:0]															
rw															

DMAn_SOURCELx contains the low base address for stream x source DMA buffer.

Bits 15:0 = **DMASOURCELx[15:0]**

Source Base Address High (DMAn_SOURCEHx) (x=0,...,3)

Address Offset: 04h - 44h - 84h - C4h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASOURCEHx[15:0]															
rw															

DMASOURCEHx contains the high base address for stream x source DMA transfer.

Bits 15:0 = **DMASOURCEHx[15:0]**

Destination Base Address Low (DMAn_DESTLx) (x=0,...,3)

Address Offset: 08h - 48h - 58h - 88h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADESTLx[15:0]															
rw															

DMADESTLx contains the low base address for stream x destination DMA buffer.

Bits 15:0 = **DMADESTLx[15:0]**

Destination Base Address High (DMAn_DESTHx) (x=0,...,3)

Address Offset: 0Ch - 4Ch - 8Ch - CCh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADESTHx[15:0]															
rw															

DMADESTHx contains the high base address for stream x destination DMA buffer.

Bits 15:0 = **DMASOURCEHx[15:0]**

Maximum Count Register (DMAn_MAXx) (x=0,...,3)

Address Offset: 10h - 50h - 8Ch - D0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAMAXx[15:0]															
rw															

This register is programmed with stream x maximum data unit count, defining the buffer size. The data unit is equal to the configured source DMA data with (byte, half-word or word). Upon enabling DMA Controller, the content of the Maximum Count Register is loaded in the Terminal Count Register.

Bits 15:0 = **DMAMAXx[15:0]**

Control Register (DMAn_CTRLx) (x=0, 1, 2)

Address Offset: 14h - 54h - 94h

Reset value: 0000h

15 14	13	12 11 10	9	8 7	6 5	4 3	2	1	0
reserved	Dir	reserved	Circular	DeSize	SoBurst	SoSize	DeInc	SoInc	Enable
-	rw	-	rw	rw	rw	rw	rw	rw	rw

The DMAn_CTRLx register is used to configure Stream x operations.

Bits 15:14 = Reserved, must be kept at reset value (0).

Bit 13 = **DIR:** *Direction transfer*
This bit is used to indicate if the peripheral is the source or the destination.
0: Peripheral is the source
1: Peripheral is the destination

Bits 12:10 = Reserved, must be kept at reset value (0).

Bit 9 = **CIRCULAR:** *Circular mode*
This bit is used to enable the DMA to operate in the circular buffer mode.
0: Normal buffer mode
1: Circular buffer mode

Bits 8:7 = **DESIZE:** *DMA to destination data width*
These bits are used to select the data width for DMA to destination data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bits 6:5 = **SOBURST:** *DMA peripheral burst size*
These bits are used to define the number of words in the peripheral burst. When the peripheral is the source, the number of (SoWidth) words read in to the FIFO before writing FIFO contents to destination. When the peripheral is the destination, the DMA interface will automatically read the correct number of source words to compile an SoBurst of the DeWith data.
00: Single
01: 4 incrementing
10: 8 incrementing
11: 16 incrementing

Bits 4:3 = **SOSIZE:** *Source to DMA data width*
These bits are used to select the data width for source to DMA data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bits 2 = **DEINC:** *Increment Current Destination Register*
This bit is used to enable the Current Destination Register after each DMA to destination data transfer.
0: Current Destination Register unchanged
1: Current Destination Register incremented

Bit 1 = **SOINC:** *Increment Current Source Register*
This bit is used to enable the Current Source Register after each source to DMA data transfer.
0: Current Source Register unchanged
1: Current Source Register incremented

Bit 0 = **ENABLE:** *DMA enable*
0: DMA disabled
1: DMA enabled

Control Register 3 (DMAn_CTRL3)

Address Offset: D4h

Reset value: 0000h

15 14	13	12	11	10	9	8 7	6 5	4 3	2	1	0
reserved	Dir	reserved	Mem2Mem	Res.	Circular	DeSize	SoBurst	SoSize	DeInc	SoInc	Enable
-	rw	-	rw	-	rw	rw	rw	rw	rw	rw	rw

The DMAn_CTRL3 register is used to configure Stream 3 operations.

Bits 15:14 = Reserved, must be kept at reset value (0).

Bit 13 = **DIR:** *Direction transfer*
This bit is used to indicate if the peripheral is the source or the destination.
0: Peripheral is the source
1: Peripheral is the destination

Bit 12 = Reserved, must be kept at reset value (0).

Bit 11 = **MEM2MEM:** *Selects memory to memory transfer*
This configures Stream 3 to operate a memory to memory transfer. When MEM2MEM is set, the DMA will disregard the DMA request connected to Stream 3, and transfer data from source to destination as fast possible until DMAn_MAX3 expires.
0: Stream3 not configured for mem to mem transfer
1: Stream3 configured for mem to mem transfer
When Stream 3 is configured as a memory-memory transfer, SOBURST relates to the source side burst length.

Bit 10 = Reserved, must be kept at reset value (0).

Bit 9 = **CIRCULAR:** *Circular mode*
This bit is used to enable the DMA to operate in circular buffer mode.
0: Normal buffer mode
1: Circular buffer mode

Bits 8:7 = **DESIZE:** *DMA to destination data width*
These bits are used to select the data width for DMA to destination data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bits 6:5 = **SOBURST:** *DMA peripheral burst size*
These bits are used to define the number of words in the peripheral burst. When the peripheral is the source, the number of (SOWIDTH) words read in to the FIFO before writing

FIFO contents to destination. When the peripheral is the destination, the DMA interface will automatically read the correct number of source words to compile an SOBURST of the DEWIDTH data.
00: Single
01: 4 incrementing
10: 8 incrementing
11: 16 incrementing

Bits 4:3 = **SOSIZE:** *Source to DMA data width*
These bits are used to select the data width for source to DMA data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bit 2 = **DEINC:** *Increment Current Destination Register*
This bit is used to enable the Current Destination Register after each DMA to destination data transfer.
0: Current Destination Register unchanged
1: Current Destination Register incremented

Bit 1 = **SOINC:** *Increment Current Source Register*
This bit is used to enable the Current Source Register after each source to DMA data transfer.
0: Current Source Register unchanged
1: Current Source Register incremented

Bit 0 = **ENABLE:** *DMA enable*
0: DMA disabled
1: DMA enabled

Current Source Address High (DMAn_SOCURRHx) (x=0,...,3)

Address Offset: 18h - 58h - 98h - D8h

Reset value: 0000h

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

The DMAn_SOCURRHx register holds the current value of the high source address pointer related to Stream x. This register is read only.

Bits 15:0 = **DMASOCURRHx[15:0]**

Current Source Address Low (DMAn_SOCURRLx) (x=0,...,3)

Address Offset: 1Ch - 5Ch - 9Ch - DCh

Reset value: 0000h

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

Then DMAn_SOCURRLx register holds the current value of the low source address pointer related to Stream x. This register is read only.

Bits 15:0 = **DMASOCURRLx[15:0]**

The value in the registers (DMAn_SOCURRLx and DMAn_SOCURRHx) is used as an AHB address in a source to DMA data transfer over the AHB bus. If the SOINC bit in the Control Register is set to ‘1’, the value in the Current Source Registers will be incremented as data are transferred from a source to the DMA. The value will be incremented at the end of the address phase of the AHB bus transfer by the transferred size value. If the SOINC bit is ‘0’, the Current Source Register will hold a same value during the whole DMA data transfer.

Current Destination Address High (DMAn_DECURRHx) (x=0,...,3)

Address Offset: 20h - 60h - A0h - E0h

Reset value: 0000h

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

DMADeCurrHiX holds the current value of the high destination address pointer related to stream X. This register is read only.

Bits 15:0 = **DMADeCurrHiX[15:0]**

Current Destination Address Low (DMAn_DECURRLx) (x=0,...,3)

Address Offset: 24h - 64h - A4h - E4h

Reset value: 0000h

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

The DMAn_DECURRLx register holds the current value of the low destination address pointer related to Stream x. This register is read only.

Bits 15:0 = **DMADECURRLx[15:0]**

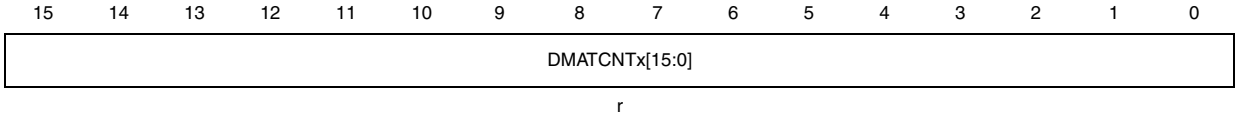
The value in the registers (DMAn_DECURRLx and DMAn_DECURRHx) is used as an AHB address in a DMA to destination data transfer over the AHB bus. If the DEINC bit in the Control Register is set to ‘1’, the value in the Current Destination Registers will be incremented as data are transferred from DMA to destination. The value will be incremented at the end of the address phase of the AHB bus transfer by the transferred size value. If

DEINC bit is ‘0’, the Current Destination Register will hold a same value during the whole DMA data transfer.

Terminal Counter Register (DMAn_TCNTx) (x=0,...,3)

Address Offset: 28h - 68h - A8h - E8h

Reset value: 0000h



The DMAn_TCNTx register contains the number of data units remaining in the current DMA transfer. The data unit is equal to the source to DMA data width (byte, half-word or word). The register value is decremented every time data is transferred to the DMA FIFO. When the Terminal Count reaches zero, the FIFO content is transferred to the Destination and the DMA transfer is finished. This is a read only register.

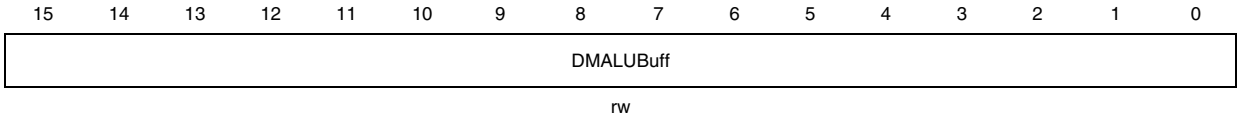
Bits 15:0 = **DMATCNTx[15:0]**

Note: DMATCntX register can be used also in Circular buffer mode, with the exception of the last buffer sweep. Once LAST flag is set, the value of DMATCntX register becomes not meaningful and should be ignored.

Last Used Buffer location X (DMAn_LUBuffX) (X=0,...,3)

Address Offset: 2Ch - 6Ch - ACh - ECh

Reset value: 0000h



DMALUBuffX is used in circular buffer mode during last buffer sweep, and it contains the circular buffer position where the last data to be used by stream X is located. The first buffer location is indicated writing 0x0000 into this register, the second with 0x0001 and so on, up to the last location which is indicated setting this register with (DMAMaxX - 1).

Bits 15:0 = **DMALUBuffX[15:0]**

Interrupt Mask Register (DMAn_MASK)

Address Offset: F0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SEM3	SEM2	SEM1	SEM0	SIM3	SIM2	SIM1	SIM0
-								rw	rw	rw	rw	rw	rw	rw	rw

The DMA Mask Register is user to select the status flag that can generate an interrupt.

Bits 15:8 = Reserved, must be kept at reset value (0).

Bit 7 = **SEM3:** *Stream 3 Error Mask*
This bit controls the generation of DMA interrupts triggered by stream 3 transfer errors events.
0: Stream 3 transfer error interrupt is masked
1: Stream 3 transfer error interrupt is enabled

Bit 6 = **SEM2:** *Stream 2 Error Mask*
This bit controls the generation of DMA interrupts triggered by stream 2 transfer errors events.
0: Stream 2 transfer error interrupt is masked
1: Stream 2 transfer error interrupt is enabled

Bit 5 = **SEM1:** *Stream 1 Error Mask*
This bit controls the generation of DMA interrupts triggered by stream 1transfer errors events.
0: Stream 1 transfer error interrupt is masked
1: Stream 1 transfer error interrupt is enabled

Bit 4 = **SEM0:** *Stream 0 Error Mask*
This bit controls the generation of DMA interrupts triggered by stream 0 transfer errors events.
0: Stream 0 transfer error interrupt is masked
1: Stream 0 transfer error interrupt is enabled

Bit 3 = **SIM3:** *Stream 3 Interrupt Mask*
This bit controls the generation of DMA interrupts triggered by stream 3 transfer end events.
0: Stream 3 transfer end interrupt is masked
1: Stream 3 transfer end interrupt is enabled

Bit 2 = **SIM2:** *Stream 2 Interrupt Mask*
This bit controls the generation of DMA interrupts triggered by stream 2 transfer end events.
0: Stream 2 transfer end interrupt is masked
1: Stream 2 transfer end interrupt is enabled

Bit 1 = **SIM1:** *Stream 1 Interrupt Mask*
This bit controls the generation of DMA interrupts triggered by stream 1 transfer end events.
0: Stream 1 transfer end interrupt is masked
1: Stream 1 transfer end interrupt is enabled

Bit 0 = **SIM0:** *Stream 0 Interrupt Mask*
This bit controls the generation of DMA interrupts triggered by stream 0 transfer end events.
0: Stream 0 transfer end interrupt is masked
1: Stream 0 transfer end interrupt is enabled

Interrupt Clear Register (DMAn_CLR)

Address Offset: F4h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SEC3	SEC2	SEC1	SEC0	SIC3	SIC2	SIC1	SIC0
-								w	w	w	w	w	w	w	w

The DMA Clear Register is used to clear the status flags. This is a write-only register.

Bits 15:8 = Reserved, must be kept at reset value (0).

Bit 7 = **SEC3:** *Stream 3 Error Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 3 transfer error event.
0: No effect
1: Clear ERR3 flag in DMAn_Status register

Bit 6 = **SEC2:** *Stream 2 Error Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 2 transfer error event.
0: No effect
1: Clear ERR2 flag in DMAn_Status register

Bit 5 = **SEC1:** *Stream 1 Error Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 1 transfer error event.
0: No effect
1: Clear ERR1 flag in DMAn_Status register

Bit 4 = **SEC0:** *Stream 0 Error Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 0 transfer error event.
0: No effect
1: Clear ERR0 flag in DMAn_Status register

Bit 3 = **SIC3:** *Stream 3 Interrupt Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 3 transfer end event.
0: No effect
1: Clear INT3 flag in DMAn_Status register

Bit 2 = **SIC2:** *Stream 2 Interrupt Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 2 transfer end event.
0: No effect
1: Clear INT2 flag in DMAn_Status register

Bit 1 = **SIC1:** *Stream 1 Interrupt Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 1 transfer end event.
0: No effect
1: Clear INT1 flag in DMAn_Status register

Bit 0 = **SIC0:** *Stream 0 Interrupt Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 0 transfer end

event.
0: No effect
1: Clear INT0 flag in DMAn_Status register

Interrupt Status Register (DMAn_STATUS)

Address Offset: F8h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ACT3	ACT2	ACT1	ACT0	ERR3	ERR2	ERR1	ERR0	INT3	INT2	INT1	INT0
-				r	r	r	r	r	r	r	r	r	r	r	r

The DMAn_Status provides status information regarding the DMA Controller. This is a read-only register.

Bits 15:12 = Reserved, must be kept cleared

Bit 11 = **ACT3:** *Data stream 3 status*
0: Data Stream 3 is not active
1: Data Stream 3 is active

Bit 10 = **ACT2:** *Data stream 2 status*
0: Data Stream 2 is not active
1: Data Stream 2 is active

Bit 9 = **ACT1:** *Data stream 1 status*
0: Data Stream 1 is not active
1: Data Stream 1 is active

Bit 8 = **ACT0:** *Data stream 0 status*
0: Data Stream 0 is not active
1: Data Stream 0 is active

Bit 7 = **ERR3:** *Data stream 3 error flag*
When a transfer error event occurs on Stream 3, this bit will be set to ‘1’ and if the SEM3 bit the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SEC3 bit in the DMAn_CLEAR register.

Bit 6 = **ERR2:** *Data stream 2 error flag*
When a transfer error event occurs on Stream 2, this bit will be set to ‘1’ and if the SEM2 bit the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SEC2 bit in the DMAn_CLEAR register.

Bit 5 = **ERR1:** *Data stream 1 error flag*
When a transfer error event occurs on Stream 1, this bit will be set to ‘1’ and if the SEM1 bit the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SEC1 bit in the DMAn_CLEAR register.

Bit 4 = **ERR0:** *Data stream 0 error flag*
When a transfer error event occurs on Stream 0, this bit will be set to ‘1’ and if the SEM0 bit the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SEC0 bit in the DMAn_CLEAR register.

Bit 3 = **INT3:** *Data stream 3 interrupt flag*
When a transfer end event occurs on Stream 3, this bit will be set to ‘1’ and if the SIM3 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC3 bit in the DMAn_CLEAR register.

Bit 2 = **INT2:** *Data stream 2 interrupt flag*
When a transfer end event occurs on Stream 2, this bit will be set to ‘1’ and if the SIM2 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC2 bit in the DMAn_CLEAR register.

Bit 1 = **INT1:** *Data stream 1 interrupt flag*
When a transfer end event occurs on Stream 1, this bit will be set to ‘1’ and if the SIM1 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC1 bit in the DMAn_CLEAR register.

Bit 0 = **INT0:** *Data stream 0 interrupt flag*
When a transfer end event occurs on Stream 0, this bit will be set to ‘1’ and if the SIM0 bit in the DMAn_MASK register has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in the SIC0 bit in the DMAn_CLEAR register.

Last Flag Register (DMAn_LAST)

Address Offset: FCh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												LAST3	LAST2	LAST1	LAST0
-												rw	rw	rw	rw

DMAn_Last controls the activation of last buffer sweep mode for the streams configured in circular buffer mode.

Bits 15:4 = Reserved, must be kept cleared.

Bit 3 = **LAST3:** *LAST buffer sweep stream 3*
This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 3 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff3 register.
0: Continuous circular buffer mode
1: Last circular buffer sweep started.

Bit 2 = **LAST2:** *LAST buffer sweep stream 2*
This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 2 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff2 register.
0: Continuous circular buffer mode
1: Last circular buffer sweep started.

Bit 1 = **LAST1:** *LAST buffer sweep stream 1*
This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 1 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff1 register.
0: Continuous circular buffer mode
1: Last circular buffer sweep started.

Bit 0 = **LAST0:** *LAST buffer sweep stream 0*
This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 0 is configured in circular mode, the corresponding data stream interrupt flag will get set when DMA uses the circular buffer location contained in DMAn_LUBuff0 register.
0: Continuous circular buffer mode
1: Last circular buffer sweep started.

8.6 DMA register map

Table 26. DMA register map

Addr. Offset	Reg. Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	DMAn_SOURCEL0	DMASOL0															
04	DMAn_SOURCEH0	DMASOH0															
08	DMAn_DESTL0	DMADEL0															
0C	DMAn_DESTH0	DMADEH0															
10	DMAn_MAX0	DMAMAX0															
14	DMAn_CTRL0	reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
18	DMAn_SOCURRH0	DMASOCURRH0															
1C	DMAn_SOCURRL0	DMASOCURRL0															
20	DMAn_DECURRH0	DMADECURRH0															
24	DMAn_DECURRL0	DMADECURRL0															
28	DMAn_TCNT0	DMATCNT0															
2C	DMAn_LUBUFF0	DMALUBuff0															
30	-	Reserved															
40	DMAn_SOURCEL1	DMASoLo1															
44	DMAn_SOURCEH1	DMASoHi1															
48	DMAn_DESTL1	DMADeLo1															
4C	DMAn_DESTH1	DMADeHi1															
50	DMAn_MAX11	DMAMax1															
54	DMAn_Ctrl1	reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
58	DMAn_SOCURRH1	DMASoCurrHi1															
5C	DMAn_SOCURRL1	DMASoCurrLo1															
60	DMAn_DECURRH1	DMADeCurrHi1															
64	DMAn_DECURRL1	DMADeCurrLo1															
68	DMAn_TCNT1	DMATCnt1															
6C	DMAn_LUBUff1	DMALUBuff1															
70	-	Reserved															
80	DMAn_SOURCEL2	DMASOL2															
84	DMAn_SOURCEH2	DMASOH2															
88	DMAn_DESTL2	DMADEL2															

Table 26. DMA register map (continued)

Addr. Offset	Reg. Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8Ch	DMAn_DESTH2	DMADEH2															
90h	DMAn_MAX2	DMAMax2															
94h	DMAn_CTRL2	reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
98h	DMAn_SOCURRH2	DMASOCURRH2															
9Ch	DMAn_SOCURRL2	DMASOCURRL2															
A0h	DMAn_DECURRH2	DMADECURRH2															
A4h	DMAn_DECURRL2	DMADECURRL2															
A8h	DMAn_TCNT2	DMATCNT2															
ACh	DMAn_LUBUFF2	DMALUBuff2															
B0h	-	Reserved															
C0h	DMAn_SOURCEL3	DMASOL3															
C4h	DMAn_SOURCEH3	DMASOH3															
C8h	DMAn_DESTL3	DMADEL3															
CCh	DMAn_DESTH3	DMADEH3															
D0h	DMAn_MAX3	DMAMAX3															
D4h	DMAn_CTRL3	reserved		Dir	res.	Mem2Mem	res	Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
D8h	DMAn_SOCURRH3	DMASOCURRH3															
DCh	DMAn_SOCURRL3	DMASOCURRL3															
E0h	DMAn_DECURRH3	DMADECURRH3															
E4h	DMAn_DECURRL3	DMADECURRL3															
E8h	DMAn_TCNT3	DMATCNT3															
ECh	DMAn_LUBUFF3	DMALUBuff3															
F0h	DMAn_MASK	Reserved								SEM3	SEM2	SEM1	SEM0	SIM3	SIM2	SIM1	SIM0
F4h	DMAn_CLR	Reserved								SEC3	SEC2	SEC1	SEC0	SIC3	SIC2	SIC1	SIC0
F8h	DMAn_STATUS	Reserved				ACT3	ACT2	ACT1	ACT0	ERR3	ERR2	ERR1	ERR0	INT3	INT2	INT1	INT0
FCh	DMAn_LAST	reserved												LAST3	LAST2	LAST1	LAST0

Refer to *Table 2 on page 17* for the base addresses.

9 Native bus arbiter (ARB)

The Native Bus Arbiter handles the arbitration between the CPU and DMA requests on the ARM native bus.

9.1 Register description

The Native Bus Arbiter registers can only be accessed when bit 29 of the CFG_PCGR1 register is set to '1' and bit 29 of the CFG_PCGRB1 register is cleared to '0' (please refer to *Section 4.3.1 on page 56*).

9.1.1 Time-Out Register (ARB_TOR)

Address: 2000 0000h
Reset value: 0000 FFFFh

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

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

Bits 31:16 = Reserved, always return '0' when read.

Bits 15:0 = **TIMEOUT[15:0]**: *DMA TimeOut Value.*
This register is a 16-bit programmable downcounter. When a DMA request wins the bus arbitration, if the DMA transaction is not completed before the counter reaches the time-out value, the DMA transaction will be stopped. When time-out elapses, the counter will reload the start value and the CPU will win the native bus arbitration until time-out elapses again. At this point, a new arbitration between CPU and DMA requests will start.
The reset value (0xFFFF) freezes the counter. To enable the counter, write any value n (0xFFFF > n > 0x0000) in the register.

9.1.2 Priority Register (ARB_PRIOR)

Address: 2000 0004h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

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

Bits 31:2 = Reserved, always return '0' when read.

Bits 1:0 = **PRIORITY[1:0]**: *DMA Priority.*
The Arbiter can handle two priority levels:
00: Only CPU can access the bus. In this case DMA requests will never be served.
01: Reserved
10: Reserved
11: DMA has the highest priority. DMA bus access will never be stopped.

9.1.3 Control Register (ARB_CTLR)

Address: 2000 000Ch
Reset value: 0000 0003h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														ABORT	res.
-														rs	-

Bits 31:2 = Reserved, always return ‘0’ when read.

Bit 1 = **ABORT**: *Native Bus Arbiter Abort Interrupt Pending Bit.*
This bit is cleared by hardware when the DMA generates a wrong address. In this case the Native Bus Arbiter internal FSM will enter “Lock” state and an Abort Interrupt Request will be issued to the CPU. In order to unlock the Native Bus Arbiter FSM, the interrupt service routine must write to ’1’ this bit.
0: DMA generated an Abort event. A corresponding Abort Interrupt Request will be issued to the CPU.
1: No DMA Abort events.
The bit is read-set. Software can only set the bit to ‘1’ , writing ‘0’ will have no effect.

Bit 0 = Reserved, must be left at reset value (1).

9.2 Native bus arbiter register map

Table 27. Native bus arbiter register map

Addr.	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2000 0000h	ARB_TOR	reserved																TIMEOUT															
2000 0004h	ARB_PRIOR	reserved																														PR IORITY	
2000 0008h	-	reserved																															
2000 000Ch	ARB_CTLR	reserved																														ABORT	reserved

10 Wake-up timer (WUT)

10.1 Introduction

This programmable free-running 16-bit timer may be used to wake-up the system from Stop mode. Two clock domains are available, the system clock and the kernel clock, one domain is completely independent from the other. The system clock is always MCLK, the kernel clock may be connected either to MCLK or to the internal RC-Oscillator. This allows the core of the wake-up timer to operate even when the STR73x is in low power mode.

When the STR73x enters Stop mode, if you keep the internal RC-Oscillator running and select the internal RC-Oscillator as Wake-up Timer kernel clock, the Wake-up Timer will continue to work. When the counter reaches the End of Count condition, it generates an internal wake-up event to the Wake-Up Unit input line 0. You can select the wake-up trigger event on Wake-up Input Line 0 between external Wake Up 0 Line and internal Wake-up Timer End Of Count event (see *Section 4.1.2 on page 54*).

The End of Count event depends on the RC-Oscillator frequency selected prior entering Stop mode. The internal prescaler allows you to program an End of Count event in a period between 500 ns and 8388 ms (with a 2 MHz RC-Oscillator frequency) and between 31 us and 524 s (with a 32 kHz RC-Oscillator frequency).

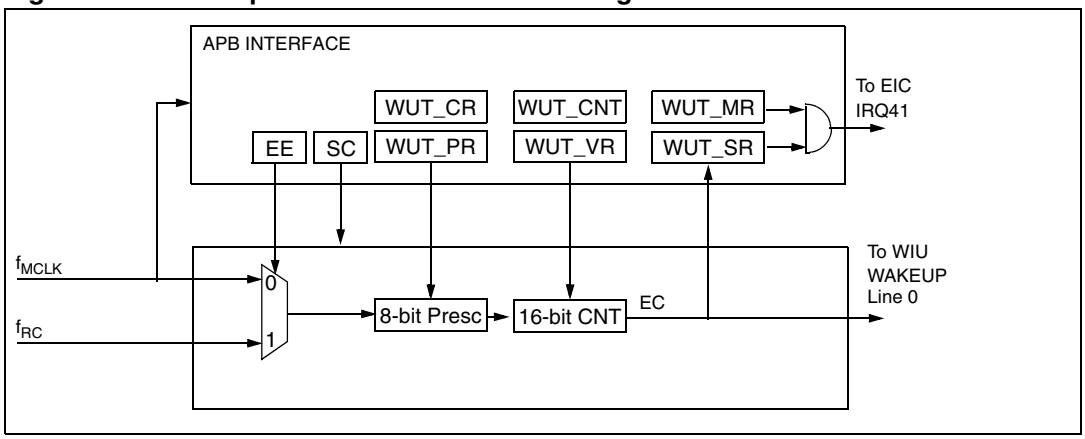
10.2 Main features

- 16-bit down counter
- 8-bit clock prescaler
- Free-running Timer mode
- End of Count interrupt generation
- Second clock source
- Operates in powerdown mode

10.3 Functional description

Figure 27 shows the functional blocks of the Wake-up Timer module. The 16-bit Counter value can be accessed through a reading of the WUT_CNT register.

Figure 27. Wake-up Timer Functional Block Diagram



10.3.1 Free-running timer mode

When enabled, the module enters in Free-running Timer mode. When in this operating mode as the SC bit in the WUT_CR register is written to '1' the WUT_VR value is loaded in the Counter and the Counter starts counting down.

When it reaches the end of count value (0000h) an End of Count interrupt is generated (EC) and the WUT_VR value is automatically re-loaded. The Counter runs until the SC bit is cleared. When the SC bit is set again, both the Counter and the Prescaler are re-loaded with the values contained in registers WUT_VR and WUT_PR respectively, so it does not restart from where it last stopped, but from a defined state without having to reset and re-program the module. On the other hand, it is not possible to change the prescaling factor on the fly since it will only effect the counter after a restart command (setting the SC bit, which generates a re-load operation).

10.4 Programming considerations

- When a write is being attempted the following method has to be used to prevent errors:
  - Read the WUT_SR to see if any of the registers are currently being written to and avoid writing to that specific register.
- When attempting to change the kernel clock by changing the EE bit in the WUT_CR the following method has to be used when the counters are running:
  - Clear the SC bit to '0', this stops the counters
  - Write to the EE bit
  - Set the SC bit to '1', this starts the counters with the new clock value
- When writing data to the Wake-up Timer it should be noted that the data is written to the registers in the "APB Interface" module immediately but it will be available in the "Kernel" module a few cycles afterwards.

  For example, if a data is written to the WUT_PR with $f_{RC} < f_{MCLK}$, then 6 f_{RC} cycles are needed to transfer the data to the counter. Similarly, if a data is written to the WUT_PR

with $f_{RC} = f_{MCLK}$, then the counter will use the new data after 3 f_{RC} (= f_{MCLK}) cycles from the time the data is written to the “APB Interface”.

- When writing to WUT_VR and WUT_PR the following sequence has to be used:
  - Clear the SC bit to ‘0’, this stops the counter
  - Write to the WUT_VR or WUT_PR
  - Set the SC bit to ‘1’, this starts the counter
  - This will ensure that the counters will operate correctly.

10.5 Register Description

The Wake-up Timer registers can not be accessed by byte.
The reserved bits can not be written and they are always read at ‘0’ (unless otherwise specified).

10.5.1 Wake-up Timer Control Register (WUT_CR)

Address Offset: 00h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													EE	SC	res
r													rw	rw	rw

Bits 15:3 = Reserved, always return ‘0’ when read.

Bit 2 = **EE**: *EXT_CK Enable bit.*
0: f_{MCLK} is used as counting clock.
1: f_{RC} signal is used as counting clock.

Bit 1 = **SC**: *Start Counting bit.*
0: the Counter is stopped. To restart it, SC setting will generate a re-loading of the Prescaler pre-load value and Timer pre-load value.
1: the Prescaler loads the Prescaler pre-load value (WUT_PR), the Counter loads the Timer pre-load value (WUT_VR) and starts counting

Bit 0 = Reserved, must be kept at its reset value (‘0’).

10.5.2 Wake-up Timer Prescaler Register (WUT_PR)

Address Offset: 04h

Reset value: 00FFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								PR[7:0]							
-								rw							

Bits 15:8 = Reserved, always return ‘0’ when read.

Bits 7:0 = **PR[7:0]**: *Prescaler value.*
The clock to Timer Counter is divided by PR[7:0]+1.
This value takes effect when the Counter starts (SC) bit is set to ‘1’.

10.5.3 Wake-up Timer Pre-load Value Register (WUT_VR)

Address Offset: 08h

Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TV[15:0]															
rw															

Bits 15:0 = **TV[15:0]**: *Timer Pre-load Value*
This value is loaded in the Timer Counter when it starts counting or an End of Count is reached. The time (us) need to reach the end of count is given by:

(PR[7:0]+1)*(TV[15:0]+1)*Tck/1000 (us)

where Tck is the Clock period measured in ns.
i.e. if CK = 20MHz the default timeout set after the system reset is 256*65536*50/1000 = 838861us.

10.5.4 Wake-up Timer Counter Register (WUT_CNT)

Address Offset: 0Ch

Reset value: FFFFh

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

Bits 15:0 = **CNT[15:0]:** *Timer Counter Value.*
The current counting value of the 16-bit Counter is available reading this register.

10.5.5 Wake-up Timer Status Register (WUT_SR)

Address Offset: 10h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved												BSY VR	BSY PR	BSYCR	EC
-												r	r	r	rc

Bits 15:4 = Reserved, always return ‘0’ when read.

Bit 3= **BSYVR**: *Reload Value Register Busy*
This bit is read only. It can be read (if it is not masked in WUT_CR) by software to see whether the WUT_VR is currently being written to.
0: WUT_VR is not currently being written to.
1: WUT_VR is currently being written to.

Bit 2= **BSYPR**: *Prescaler Register Busy*
This bit is read only. It can be read (if it is not masked in WUT_CR) by software to see whether the WUT_PR is currently being written to.
0: WUT_PR is not currently being written to.
1: WUT_PR is currently being written to.

Bit 1= **BSYCR**: *Control Register Busy*
This bit is read only. It can be read (if it is not masked in WUT_CR) by software to see whether the WUT_CR is currently being written to.
0: WUT_CR is not currently being written to.
1: WUT_CR is currently being written to.

Bit 0 = **EC**: *End of Count pending bit.*
This bit can be set only by hardware and must be reset in software by writing '0' to this address.
0: No End of Count has occurred
1: The End of Count has occurred

10.5.6 Wake-up Timer Mask Register (WUT_MR)

Address Offset: 14h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														BSY MSK	ECM
-														rw	rw

Bits 15:2 = Reserved, always return '0' when read.

Bit 1= **BSYMSK**: *Busy Bit Mask*
This bit determines whether the software can read the busy bits in the WUT_SR. After reset the software can not read the busy bits until this bit is set to '1'.
0: Busy bits are masked.
1: Busy bits are not masked.

Bit 0 = **ECM**: *End of Count Mask bit.*
0: End of Count interrupt request is disabled
1: End of Count interrupt request is enabled

10.6 WUT register map

Table 28. Wake-up timer peripheral register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	WUT_CR	reserved													EE	SC	WE
04h	WUT_PR	reserved								PR(7:0)							
08h	WUT_VR	TV(15:0)															
0Ch	WUT_CNT	CNT(15:0)															
10h	WUT_SR	reserved												BSY VR	BSY PR	BSY CR	EC
14h	WUT_MR	reserved														BSY MSK	ECM

Refer to *Table 2 on page 17* for the base address.

11 Real time clock (RTC)

11.1 Introduction

The RTC provides a set of continuously running counters which can be used, with suitable software, to provide a clock-calendar function. The counter values can be written to set the current time/date of the system.

The RTC includes an APB slave interface, to provide access by word to internal 32-bit registers; this interface is disconnected from the APB bus when the main power supply is removed.

11.2 Main features

- Fixed divider prescaler: 1/64
- Programmable prescaler: division factor up to 2^{20}
- 32-bit programmable counter for long term measurement
- External clock input (must be at least 4 times slower than MCLK clock)
- 4 dedicated maskable interrupt lines:
  - Alarm interrupt, for generating a software programmable alarm interrupt
  - Seconds interrupt, for generating a periodic interrupt signal with a programmable period length (up to 1 sec.)
  - Overflow interrupt, to detect when the internal programmable counter rolls over to zero
  - Global interrupt: a logical OR function of all the above, to allow a single interrupt channel to manage all the interrupt sources

11.3 Functional description

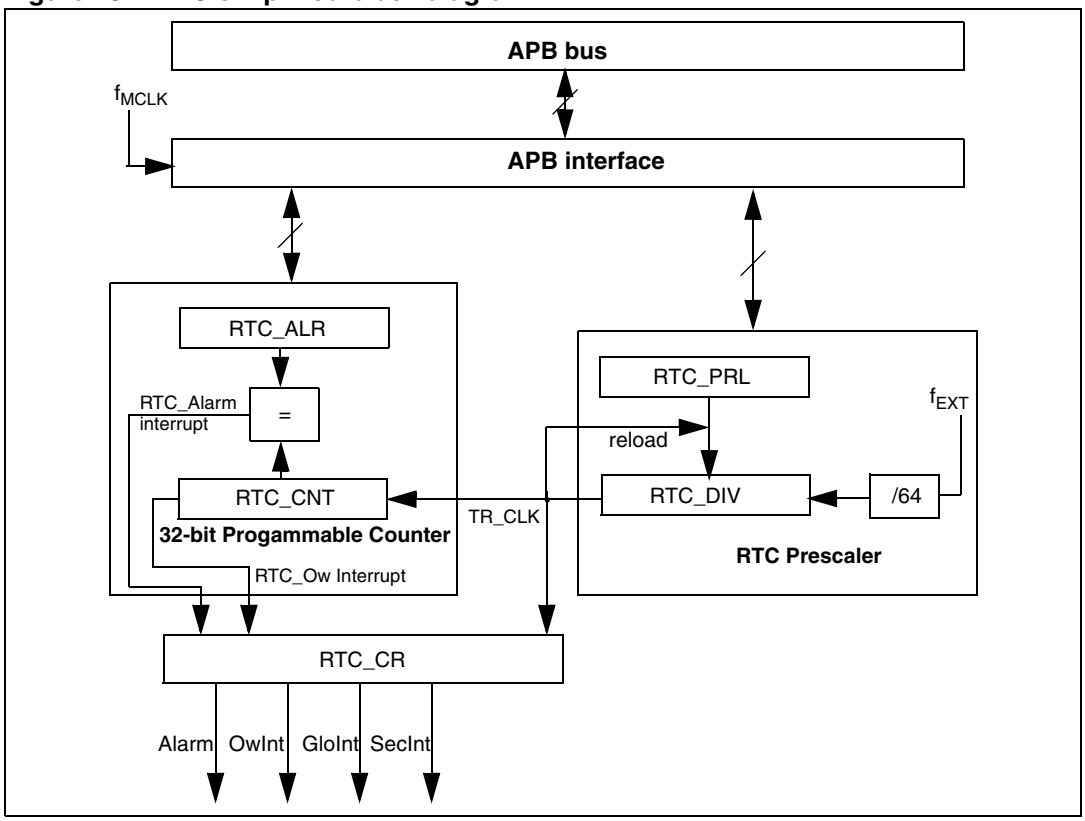
11.3.1 Overview

The RTC consists of two main units (see *Table 28: RTC simplified block diagram*), the first one (APB Interface) is used to interface the APB bus. This unit also contains a set of 16-bit registers, synchronous to MCLK and accessible from the APB bus in read or write mode (for more details refer to Register description *Section 11.4*). The APB interface is clocked by MCLK in order to interface with the APB bus.

The other unit (RTC Core) consists of a chain of programmable counters made of 2 main blocks. The first block is the RTC prescaler block which generates the RTC time base TR_CLK which can be programmed to have a period of up 1 second. It includes a fixed divider prescaler (1/64), connected to f_{EXT} and a 20-bit programmable divider (RTC Prescaler). Every TR_CLK period, the RTC generates an interrupt (SecInt) if it is enabled in the RTC_CRH control register. The second block is a 32-bit programmable counter that can be initialized to the current system time. The system time is incremented at the TR_CLK rate and compared with a programmable date (stored in the RTC_ALR register) in order to generate an alarm interrupt, if enabled in RTC_CRH control register.

Note: The RTC does not have a separate clock, reset or supply input, consequently the time information is reset when the STR73x is reset.

Figure 28. RTC simplified block diagram



11.3.2 Free-running mode

After Power-on reset, the peripheral enters free-running mode if the corresponding bit in the CFG_PCGR0 register is set. In this operating mode, the RTC Prescaler and the Programmable counter start counting. Interrupt flags are activated too but, since interrupt signals are masked, there is no interrupt generation. Interrupt signals must be enabled by setting the appropriate bits in the RTC_CRH register. In order to avoid spurious interrupt generation it is recommended to clear old interrupt requests before enabling them.

11.3.3 RTC flag assertion

The RTC Second Interrupt Request (SIR) is asserted at each RTC Core clock cycle before the update of the RTC Counter.

The RTC Overflow Interrupt Request (OWIR) is asserted at the last RTC Core clock cycle before the counter reaches the 0x0000 value.

The RTC_Alarm and RTC Alarm Interrupt Request (AIR) , see *Figure 29*, are asserted at the last RTC Core clock cycle before the counter reaches the RTC Alarm value stored in the Alarm register increased by one (RTC_ALR + 1). To set the RTC Alarm value, you must be sure that this write is synchronized with the RTC Second flag. For this purpose two alternative methods can be used:

- Use the RTC Alarm interrupt and update the RTC Alarm and/or RTC Counter registers in the RTC interrupt service routine.
- Wait for the SIR flag until it is set and then update the RTC Alarm and/or RTC Counter registers.

Note: If RTC interrupts are used during Run, Slow, WFI or LPWFI modes the RTC clock must be at least 4 times slower than MCLK clock.

Figure 29. RTC second and alarm waveform example with PR=0003, ALARM=00004

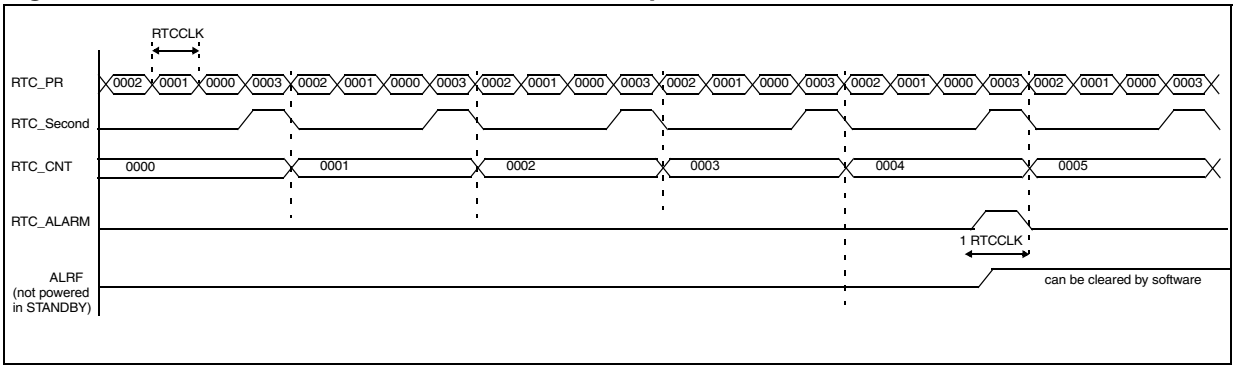
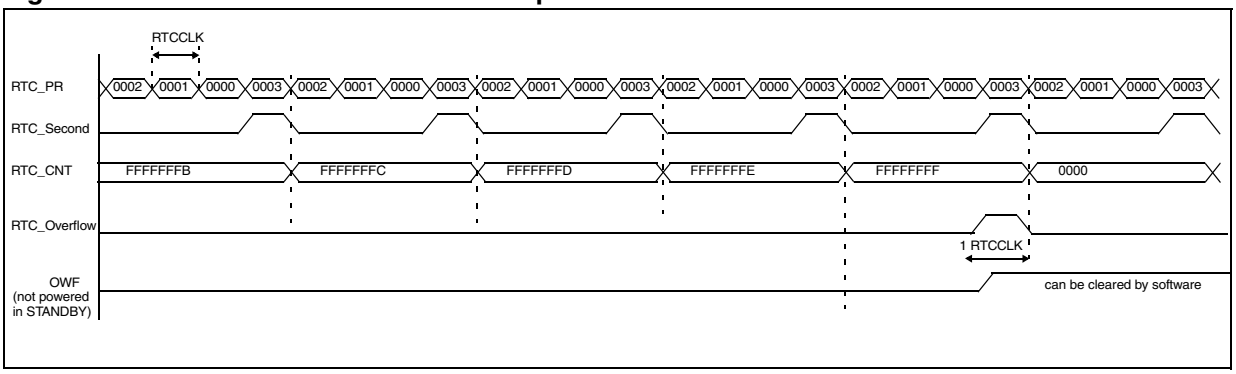


Figure 30. RTC Overflow waveform example with PR=0003



11.3.4 Configuration mode

To write in RTC_PRL, RTC_CNT, RTC_ALR registers, the peripheral must enter Configuration mode. This is done setting the CNF bit in the RTC_CRL register.

In addition, writing to any RTC register is only enabled if the previous write operation is finished. To enable the software to detect this situation, the RTOFF status bit is provided in the RTC_CRL register to indicate that an update of the registers is in progress. A new value can be written to the RTC counters only when the RTOFF status bit value is ’1’.

Configuration Procedure:

1. Poll RTOFF, wait until its value goes to ‘1’
2. Set CNF bit to enter configuration mode
3. Write to one or more RTC registers
4. Clear CNF bit to exit configuration mode

The write operation only executes when the CNF bit is cleared and it takes at least three f_{EXT} cycles to complete.

11.4 Register description

The RTC registers cannot be accessed by byte. The reserved bits can not be written and they are always read as ‘0’.

11.4.1 RTC Control Register High (RTC_CRH)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												GEN	OWEN	AEN	SEN
-												rw	rw	rw	rw

These bits are used to mask interrupt requests. Note that at reset all interrupts are disabled, so it is possible to write to the RTC registers to ensure that no interrupt requests are pending after initialization. It is not possible to write RTC_CRH register when the peripheral is completing a previous write operation (flagged by RTOFF=0, see *Configuration mode on page 139*).

The functions of the RTC are controlled by this control register. Some bits must be written using a specific configuration procedure (see *Configuration mode on page 139*).

Bits 15:4 = Reserved

Bit 3 = **GEN**: *Global interrupt ENable*
0: Global interrupt is masked.
1: Global interrupt is enabled.

Bit 2 = **OWEN**: *Overflow interrupt ENable*
0: Overflow interrupt is masked.
1: Overflow interrupt is enabled.

Bit 1 = **AEN**: *Alarm interrupt ENable*
0: Alarm interrupt is masked.
1: Alarm interrupt is enabled.

Bit 0 = **SEN**: *Second interrupt ENable*
0: Second interrupt is masked.
1: Second interrupt is enabled.

11.4.2 RTC Control Register Low (RTC_CRL)

Address Offset: 04h
Reset value: 0020h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RTOFF	CNF	GIR	OWIR	AIR	SIR
-										r	rw	rc	rc	rc	rc

The functions of the RTC are controlled by this control register. It is not possible to write RTC_CR register when the peripheral is completing a previous write operation (flagged by RTOFF=0, see *Configuration mode on page 139*).

Bits 15:6 = Reserved

Bit 5 = **RTOFF**: *RTC operation OFF*
With this bit the RTC reports the status of the last write operation performed on its registers, indicating if it has been completed or not. If its value is '0' then it is not possible to write to any of the RTC registers. This bit is read only.
0: Last write operation on RTC registers is still ongoing.
1: Last write operation on RTC registers terminated.

Bit 4 = **CNF**: *Configuration Flag*
This bit must be set by software to enter configuration mode so as to allow new values to be written in the RTC_CNT, RTC_ALR or RTC_PRL registers. The write operation is only executed when, the CNF bit is reset by software after has been set.
0: Exit configuration mode (start update of RTC registers).
1: Enter configuration mode.

Bit 3 = **GIR**: *Global Interrupt Request*
This bit contains the status of global interrupt request signal, which is goes high when at least one of the other interrupt lines is active. When this bit is set, the corresponding interrupt will be generated only if GEN bit is set. The GIR bit can be set only by hardware and can be cleared only by software, while writing '1' will left it unchanged.
0: GloInt interrupt condition not met.
1: GloInt interrupt request pending.

Bit 2 = **OWIR**: *Overflow Interrupt Request*
This bit stores the status of periodic interrupt request signal (*RTC_OwIT*) generated by the overflow of the 32-bit programmable counter. When this bit is at '1', the corresponding interrupt will be generated only if OWEN bit is set to '1'. OWEN bit can be set at '1' only by hardware and can be cleared only by software, while writing '1' will left it unchanged.
0: Overflow interrupt condition not met.
1: Overflow interrupt request pending.

Bit 1 = **AIR**: *Alarm Interrupt Request*
This bit contains the status of periodic interrupt request signal (*RTC_AlarmIt*) generated by the 32 bit programmable counter when the threshold set in RTC_ALR register is reached. When this bit is at '1', the corresponding interrupt will be generated only if the AEN bit is set to '1'. The AIR bit can be set at '1' only by hardware and can be cleared only by software, while writing '1' will left it unchanged.
0: Alarm interrupt condition not met.
1: Alarm interrupt request pending.

Bit 0 = **SIR**: *Second Interrupt Request*
This bit contains the status of second interrupt request signal (*RTC_SecIt*) generated by the

overflow of the 20-bit programmable prescaler which increments the RTC counter. Hence this Interrupt provides a periodic signal with a period corresponding to the resolution programmed for the RTC counter (usually one second). When this bit is at '1', the corresponding interrupt will be generated only if the SEN bit is set to '1'. The SIR bit can be set at '1' only by hardware and can be cleared only by software, while writing '1' will left it unchanged.
0: 'Second' interrupt condition not met.
1: 'Second' interrupt request pending.

Note: 1 Any interrupt request remains pending until the appropriate RTC_CRL request bit is reset by software, indicating that the interrupt request has been granted.

2 At reset the interrupts are disabled, no interrupt requests are pending and it is possible to write the RTC registers.

3 The OWIR, AIR and SIR bits are not updated when the system clock MCLK is not running (for example when the STR73x is in Stop Mode.

4 The SIR, AIR, OWIR and GIR bits can only be set by hardware and cleared only by software.

11.4.3 RTC Prescaler Load Register High (RTC_PRLH)

Address Offset: 08h
Read/Write (see *Configuration mode on page 139*)
Reset value: 0000h

The Prescaler Load registers keep the period counting value of the RTC prescaler. They are write protected by the RTOFF bit in the RTC_CRL register, write operation is allowed if RTOFF value is '1'.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												PRL(19:16)			
-												rw			

Bits 15:4 = Reserved

Bits 3:0 = **PRL[19:16]:** *RTC Prescaler Reload value high*

These bits are used to define the counter clock frequency according to the following formula:

$f_{TR\_CLK} = f_{RTC}/64/2$; if PRL[19:0] =0

$f_{TR\_CLK} = f_{RTC}/64/(PRL[19:0]+1)$; if PRL[19:0] > 0

11.4.4 RTC Prescaler Load Register Low (RTC_PRLL)

Address Offset: 0Ch
Read/Write (see *Configuration mode on page 139*)
Reset value: 8000h

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

Bits 15:0 = **PRL[15:0]:** *RTC Prescaler Reload value low*

These bits are used to define the counter clock frequency according to the following formula:

$f_{TR\_CLK} = f_{RTC}/64/2$; if PRL[19:0] =0

$f_{TR\_CLK} = f_{RTC}/64/(PRL[19:0]+1)$; if PRL[19:0] > 0

Note: The reset value sets the TR_CLK signal period to 1 sec for an input clock of 32 kHz.

11.4.5 RTC Prescaler Divider Register High (RTC_DIVH)

Address Offset: 10h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												DIV[19:16]			
-												r			

Bits 15:4 = Reserved

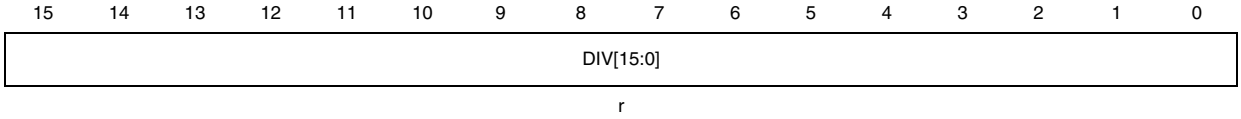
Every period of TR_CLK the counter inside RTC prescaler divider is reloaded with the value stored in the RTC_PRL register. To get an accurate time measurement it is possible to read the current value of the prescaler counter, stored in the RTC_DIV register, without stopping

it. This register is read only and it is reloaded by hardware after any change in RTC_PRL or RTC_CNT registers.

Bits 3:0 = **DIV[19:16]:** *RTC Clock Divider High*

11.4.6 RTC Prescaler Divider Register Low (RTC_DIVL)

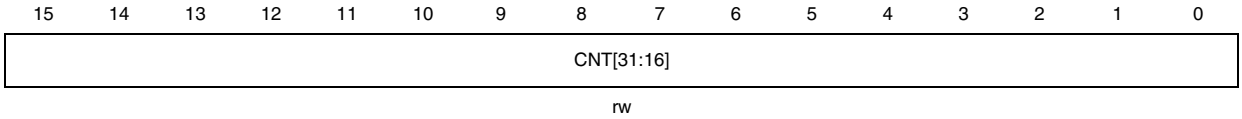
Address Offset: 14h
Reset value: 8000h



Bits 3:0 = **DIV[15:0]:** *RTC Clock Divider Low*

11.4.7 RTC Counter Register High (RTC_CNTH)

Address Offset: 18h
Reset value: 0000h

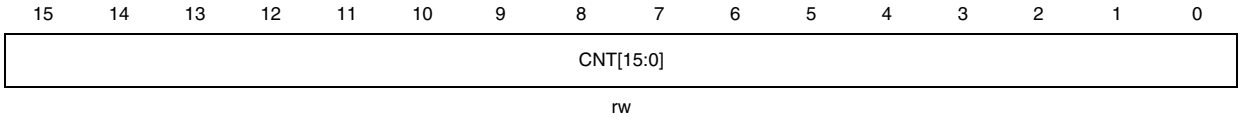


The RTC core has one 32-bit programmable counter, accessed through 2 16-bit registers; the count rate is based on the TR_Clock time reference, generated by the prescaler. RTC_CNT registers keep the counting value of this counter. They are write protected by bit RTOFF in the RTC_CRL register, write operation is allowed if RTOFF value is ‘1’. A write operation on the upper (RTC_CNTH) or lower (RTC_CNTL) registers directly loads the corresponding programmable counter and reloads the RTC Prescaler. When reading, the current value in the counter (system date) is returned.

Bits 15:0 = **CNT[31:16]:** *RTC Counter High*
Reading RTC_CNTH register, the current value of the high part of RTC Counter register is returned. To write this register it is required to enter configuration mode using the RTOFF bit in the RTC_CRL register.

11.4.8 RTC Counter Register Low (RTC_CNTL)

Address Offset: 1Ch
Reset value: 0000h



Bits 15:0 = **CNT[15:0]:** *RTC Counter Low*
Reading RTC_CNTL register, the current value of the lower part of RTC Counter register is returned. To write this register it is required to enter configuration mode using the RTOFF bit in the RTC_CR register.

11.4.9 RTC Alarm Register High (RTC_ALRH)

Address Offset: 20h
Write only (see *Configuration mode on page 139*)
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ALARM[31:16]															
w															

When the programmable counter reaches the 32-bit value stored in the RTC_ALR registers, an alarm is triggered and the RTC_alarmIT interrupt request is generated. This register is write protected by the RTOFF bit in the RTC_CRL register, write operation is allowed if the RTOFF value is '1'.

Bits 15:0 = **ALARM[31:16]:** *RTC Alarm High*
The high part of alarm time is written by software in this register. To write this register it is required to enter configuration mode using the RTOFF bit in the RTC_CRL register.

11.4.10 RTC alarm register Low (RTC_ALRL)

Address Offset: 24h
Write only (see *Configuration mode on page 139*)
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ALARM[15:0]															
w															

Bits 15:0 = **ALARM[15:0]:** *RTC Alarm Low*
The low part of alarm time is written by software in this register. To write this register it is required to enter configuration mode using the RTOFF bit in the RTC_CRL register.

11.5 RTC register map

RTC registers are mapped as 16-bit addressable registers as described in the table below:

Table 29. RTC register map

Address Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	RTC_CRH	---												GEN	OWEN	AEN	SEN
04h	RTC_CRL	---										RTOFF	CNF	GIR	OWIR	AIR	SIR
08h	RTC_PRLH	---												PRL[19:16]			
0Ch	RTC_PRLL	PRL[15:0]															
10h	RTC_DIVH	---												DIV[19:16]			
14h	RTC_DIVL	DIV[15:0]															

Table 29. RTC register map (continued)

Address Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18h	RTC_CNTH	CNT[31:16]															
1Ch	RTC_CNTL	CNT[15:0]															
20h	RTC_ALRH	ALARM[31:16]															
24h	RTC_ALRL	ALARM[15:0]															

See *Table 2 on page 17* for the base address

12 Watchdog timer (WDG)

12.1 Introduction

The Watchdog Timer peripheral can be used as free-running timer or as Watchdog to resolve processor malfunctions due to hardware or software failures.

12.2 Main Features

- 16-bit down counter
- 8-bit clock prescaler
- Safe reload sequence
- Free-running timer mode
- End of Count interrupt generation
- Second clock source

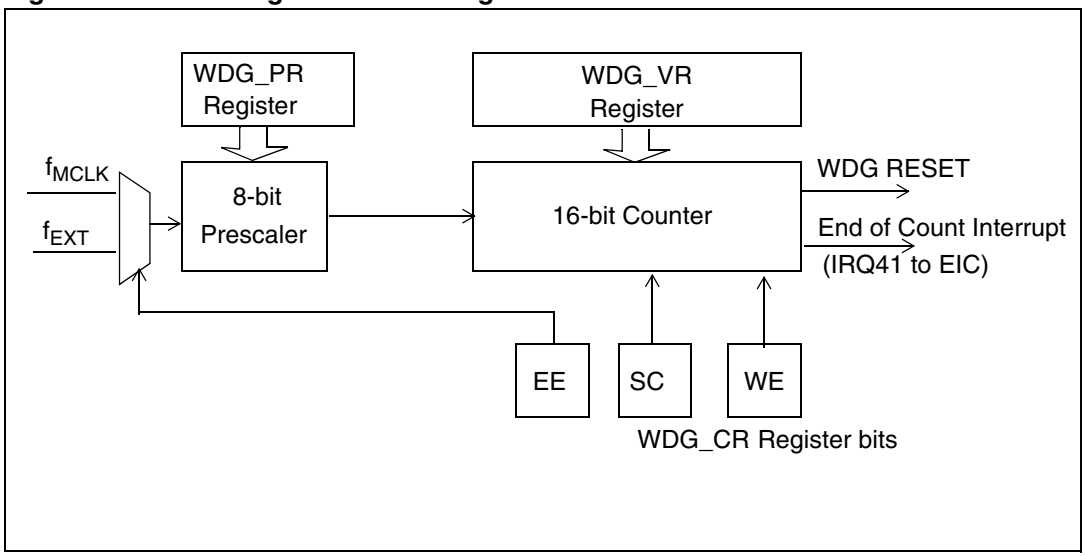
12.3 Functional description

Figure 31 shows the functional blocks of the Watchdog Timer module. The module can work as a Watchdog or as a Free-running Timer. In both modes the 16-bit Counter value can be accessed by reading the WDG_CNT register.The WDGEN comes from a device input pin or from an internal hardware signal (refer to device specification).

12.3.1 Free-running timer mode

If the WDGEN signal value straight after the reset has elapsed is low and the WE bit of WDG_CR register is not written to ‘1’ by software, the peripheral enters Free-running Timer mode. WDGEN signal, depending on device implementation, can be tied directly to a logic level ‘1’ or can be available on a specific external pin (named HW0SW1) to allow the user to enable or not the Watchdog Timer before exiting from reset.
When in this operating mode as the SC bit of WDG_CR register is written to ‘1’ the WDG_VR value is loaded in the Counter and the Counter starts counting down.

Figure 31. Watchdog timer block diagram



When it reaches the end of count value (0000h) an End of Count interrupt is generated (EC_int) and the WDG_VR value is re-loaded. The Counter runs until the SC bit is cleared. If the SC bit is set again, both the Counter and the Prescaler are re-loaded with the values contained in registers WDTVR and WDTPR respectively, so it does not restart from where it last stopped, but from a defined state without having to reset and re-program the module. On the other hand, it is not possible to change the prescaling factor on-the-fly since it will only effect the counter after a restart command (setting the SC bit, which generates a re-load operation).

The clock input signal can be either the system clock (f_{MCLK}) or f_{EXT} which must have a period at least four times longer than the system clock period. This allows the watchdog time base to be independent of the system clock which could change dynamically depending on the operating mode (run mode, low power mode, etc.).

12.3.2 Watchdog mode

If the WDGEN signal value straight after the reset has elapsed is high or theWE bit of WDG_CR register is written to ‘1’ by software, the peripheral enters Watchdog mode. This operating mode can not be changed by software (the SC bit has no effect and WE bit cannot be cleared).
As the peripheral enters in this operating mode, the WDG_VR value is loaded in the Counter and the Counter starts counting down. When it reaches the end of count value (0000h) a system reset signal is generated (WDG RESET).

If a sequence of two consecutive values (0xA55A and 0x5AA5) is written in the WDG_KR register see *Section 12.4*, the WDG_VR value is re-loaded in the Counter, the End of count can be prevented.

12.4 Register description

The Watchdog Timer registers can not be accessed by byte.
The reserved bits can not be written and they are always read at ‘0’.

12.4.1 WDG control register (WDG_CR)

Address Offset: 00h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													EE	SC	WE
-													rw	rw	rw

Bits 15:3 = Reserved, must be kept at reset value (0).

Bit 2 = **EE**: *EXT_CK Enable bit*
0: f_{MCLK} is used as counter clock.
This bit can not be written when Watchdog mode is enabled (WE bit = 1)
1: f_{EXT} is used as counter clock. f_{EXT} period must be at least 4 times CK period.

Bit 1 = **SC**: *Start Counting bit*
0: The counter is stopped.
1: The counter loads the Timer pre-load value and starts counting
These functions are permitted only in Timer Mode (WE bit = 0).

Bit 0 = **WE**: *Watchdog Enable bit*
0: Timer Mode is enabled
1: Watchdog Mode is enabled
This bit can't be reset by software.

If the external WDGEN signal is high the WE bit is written to '1' by hardware as soon the reset has elapsed else can write to '1' by software but cannot be cleared.

When WE bit is high, SC bit has no effect.

12.4.2 WDG prescaler register (WDG_PR)

Address Offset: 04h

Reset value: 00FFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
-								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 = *Reserved, must be kept at reset value (0)*

Bit 7:0 = **PR[7:0]**: *Prescaler value*
The clock to Timer Counter is divided by PR[7:0]+1.
This value takes effect when Watchdog mode is enabled (WE bit is set) or the re-load sequence occurs or the Counter starts (SC) bit is set in Timer mode.

12.4.3 WDG preload value register (WDG_VR)

Address Offset: 08h

Reset value: FFFFh

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

TV15	TV14	TV13	TV12	TV11	TV10	TV9	TV8	TV7	TV6	TV5	TV4	TV3	TV2	TV1	TV0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **TV[15:0]**: *Timer Pre-load Value*
This value is loaded in the Timer Counter when it starts counting or a re-load sequence occurs or an End of Count is reached. The time (µs) need to reach the end of count is given by:

(PR[7:0]+1)*(TV[15:0]+1)*t_{CLK}/1000 (µs)

where t_{CLK} is the Clock period measured in ns.
For example, if CLK = 20 MHz, the default timeout set after the system reset is 256*65535*50/1000 = 838800 µs.

12.4.4 WDG counter register (WDG_CNT)

Address Offset: 0Ch

Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 = **CNT[15:0]:** *Timer Counter Value*
The current counting value of the 16-bit Counter is available reading this register.

12.4.5 WDG status register (WDG_SR)

Address Offset: 10h

Reset value: 0000h

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

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **EC**: *End of Count pending bit*
0: no End of Count has occurred
1: the End of Count has occurred
In Watchdog Mode (WE = 1) this bit has no effect.
This bit can be set only by hardware and must be reset by software.

12.4.6 WDG mask register (WDG_MR)

Address Offset: 14h

Reset value: 0000h

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

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **ECM**: *End of Count Mask bit*
0: End of Count interrupt request is disabled
1: End of Count interrupt request is enabled

12.4.7 WDG key register (WDG_KR)

Address Offset: 18h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
K15	K14	K13	K12	K11	K10	K9	K8	K7	K6	K5	K4	K3	K2	K1	K0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **K[15:0]:** *Key Value*
When Watchdog Mode is enabled, writing in this register two consecutive values (A55A, 5AA5) the Counter is initialized to TV[15:0] value and the Prescaler value in WTDPR register take effect. Any number of instructions can be executed between the two writes.
If Watchdog Mode is disabled (WE = 0) a write to this register has no effect.
This register returns the value 0000h when read.

12.5 WDG register map

Table 30. Watchdog timer register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	WDG_CR	Reserved														SC	WE
04h	WDG_PR	Reserved								PR(7:0)							
08h	WDG_VR	TV(15:0)															
0Ch	WDG_CNT	TV(15:0)															
10h	WDG_SR	Reserved															EC
14h	WDG_MR	Reserved															MEC
18h	WDG_KR	K[15:0]															

See *Table 2 on page 17* for the base address

13 Timebase timer (TB)

Three TB Timers are implemented in the STR73x device. Each of them consists of a 16-bit down counter with an 8-bit clock prescaler which can be used as free-running timer for internal time base generation.

The f_{EXT} clock source can be selected by software (in place of f_{MCLK}) to obtain a time base independent of the system clock frequency (usually dependent on the selected system configuration mode).

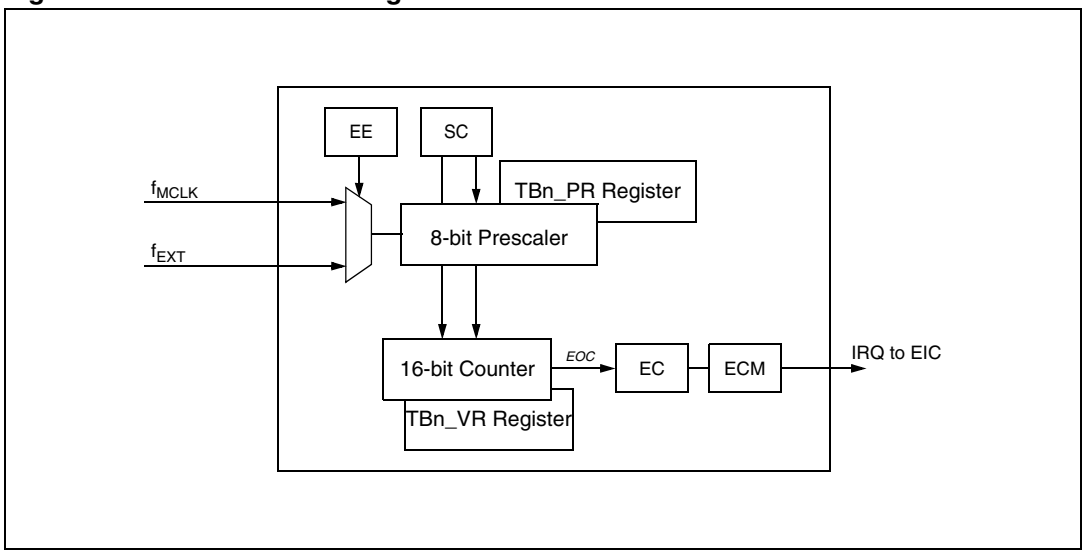
13.1 Main features

- 16-bit down counter
- 8-bit clock prescaler
- Free-running timer mode
- End of Count interrupt generation
- Dual clock source

13.2 Functional description

Figure 32 shows the functional blocks of the TB Timer module. The 16-bit Counter value can be accessed by reading the TBn_CNT register.

Figure 32. TB timer block diagram



13.2.1 Free-running timer mode

When enabled, the module enters Free-running Timer mode. In this mode, when the SC bit in the TBn_CR register is set by software, the TBn_VR value is loaded in the Counter and the Counter starts counting down.

When it reaches the end of count value (0000h) an End of Count interrupt is generated (EC) and the TBn_VR value is automatically re-loaded. The Counter runs until the SC bit is cleared. When the SC bit is set again, both the Counter and the Prescaler are re-loaded with the values contained in registers TBn_VR and TBn_PR respectively, so it does not restart from where it last stopped, but from a defined state without requiring the application to reset and re-program the module. On the other hand, it is not possible to change the prescaling factor on-the-fly since it will only effect the counter after a restart command (SC bit setting which generates a re-load operation).

The clock input signal can be either the system clock (f_{MCLK}) or f_{EXT}, which must have a period at least four times longer than the system clock period. This allows the time base to be independent of the system clock which could change dynamically depending on the operating mode (run mode, low power mode, etc.).

13.3 Register description

The TB Timer registers can not be accessed by byte.
The reserved bits can not be written and they are always read at ‘0’ (unless otherwise specified).

TB Timer Control Register (TBn_CR)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													EE	SC	res.
-													rw	rw	rw

Bits 15:3 = Reserved, must be kept at reset value (0).

Bit 2 = **EE**: *External Clock Source Enable bit.*
0: CK is used as counting clock.
1: The EXT_CK signal is used as counting clock. EXT_CK period must be at least 4 times CK period.

Bit 1 = **SC**: *Start Counting bit.*
0: the Counter is stopped. To restart it, SC setting will generate a re-loading of the Prescaler pre-load value and Timer pre-load value.
1: the Prescaler loads the Prescaler pre-load value (TBn_PR), the Counter loads the Timer pre-load value (TBn_VR) and starts counting

Bit 0 = Reserved, must be kept at reset value (0).

TB Timer Prescaler Register (TBn_PR)

Address Offset: 04h
Reset value: 00FFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
-								rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:8 = Reserved, must be kept at reset value (0).

Bits 7:0 = **PR[7:0]**: *Prescaler value.*
The clock to the Timer Counter is divided by PR[7:0]+1.
This value takes effect when the Counter Starts (SC) bit is put to ‘1’.

TB Timer Pre-load Value Register (TBn_VR)

Address Offset: 08h
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TV15	TV14	TV13	TV12	TV11	TV10	TV9	TV8	TV7	TV6	TV5	TV4	TV3	TV2	TV1	TV0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **TV[15:0]**: *Timer Pre-load Value.*
This value is loaded in the Timer Counter when it starts counting or an End of Count is reached. The time (µs) needed to reach the end of count is given by:

(PR[7:0]+1)*(TV[15:0]+1)*Tck/1000 (µs)

where Tck is the Clock period measured in ns.
For example, if CK = 20 MHz the default timeout set after the system reset is 256*65536*50/1000 = 838861 µs.

TB Timer Counter Register (TBn_CNT)

Address Offset: 0Ch
Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 15:0 = **CNT[15:0]:** *Timer Counter Value.*
The current counting value of the 16-bit Counter is available reading this register.

TB Timer Status Register (TBn_SR)

Address Offset: 10h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															EC
-															rc

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **EC**: *End of Count pending bit.*
0: No End of Count has occurred
1: End of Count has occurred
This bit can be set only by hardware and must be reset by software.

TB Timer Mask Register (TBn_MR)

Address Offset: 14h
Reset value: 0000h

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

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **ECM**: *End of Count Mask bit.*
0: End of Count interrupt request is disabled.
1: End of Count interrupt request is enabled.

13.4 TB register map

Table 31. TB timer peripheral register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	TBn_CR	reserved													EE	SC	res.
04h	TBn_PR	reserved								PR(7:0)							
08h	TBn_VR	TV(15:0)															
0Ch	TBn_CNT	CNT(15:0)															
10h	TBn_SR	reserved															EC
14h	TBn_MR	reserved															ECM

See *Table 2 on page 17* for the base address

14 Timer (TIM)

14.1 Introduction

Each TIM timer consists of a 16-bit counter driven by a programmable prescaler.

It may be used for a variety of purposes, including pulse length measurement of up to two input signals (input capture) or generation of up to two output waveforms (output compare and PWM).

Pulse lengths and waveform periods can be modulated from a very wide range using the timer prescaler.

The STR73x DMA controllers can be used transfer data to/from the TIM timers and memory.

14.2 Main features

- Programmable prescaler: f_{MCLK} divided from 1 to 256, Prescaler register (0 to 255) value +1.
- Overflow status flag and maskable interrupts
- External clock input (must be at least 4 times slower than the MCLK clock speed) with the choice of active edge.
- Selectable External clock source f_{EXT} or ICAPA pin. Configured by CFG_TIMSR register
- Output compare functions with
- 2 dedicated 16-bit registers
- 2 dedicated programmable signals
- 2 dedicated status flags
- 2 dedicated interrupt flags.
- Input capture functions with
- 2 dedicated 16-bit registers
- 2 dedicated active edge selection signals
- 2 dedicated status flags
- 2 dedicated interrupt flags.
- Pulse width modulation mode (PWM)
- One pulse mode (OPM)
- PWM input mode
- Timer global interrupt (5 internally ORed sources)
- ICIA: Timer Input capture A interrupt
- ICIB: Timer Input capture B interrupt
- OCIA: Timer Output compare A interrupt
- OCIB: Timer Output compare B interrupt
- TOI: Timer Overflow interrupt.
- 1 channel DMA support

The block diagram is shown in *Figure 33*.

14.3 Functional description

14.3.1 Counter

The principal block of the Programmable Timer is a 16-bit counter and its associated 16-bit registers.

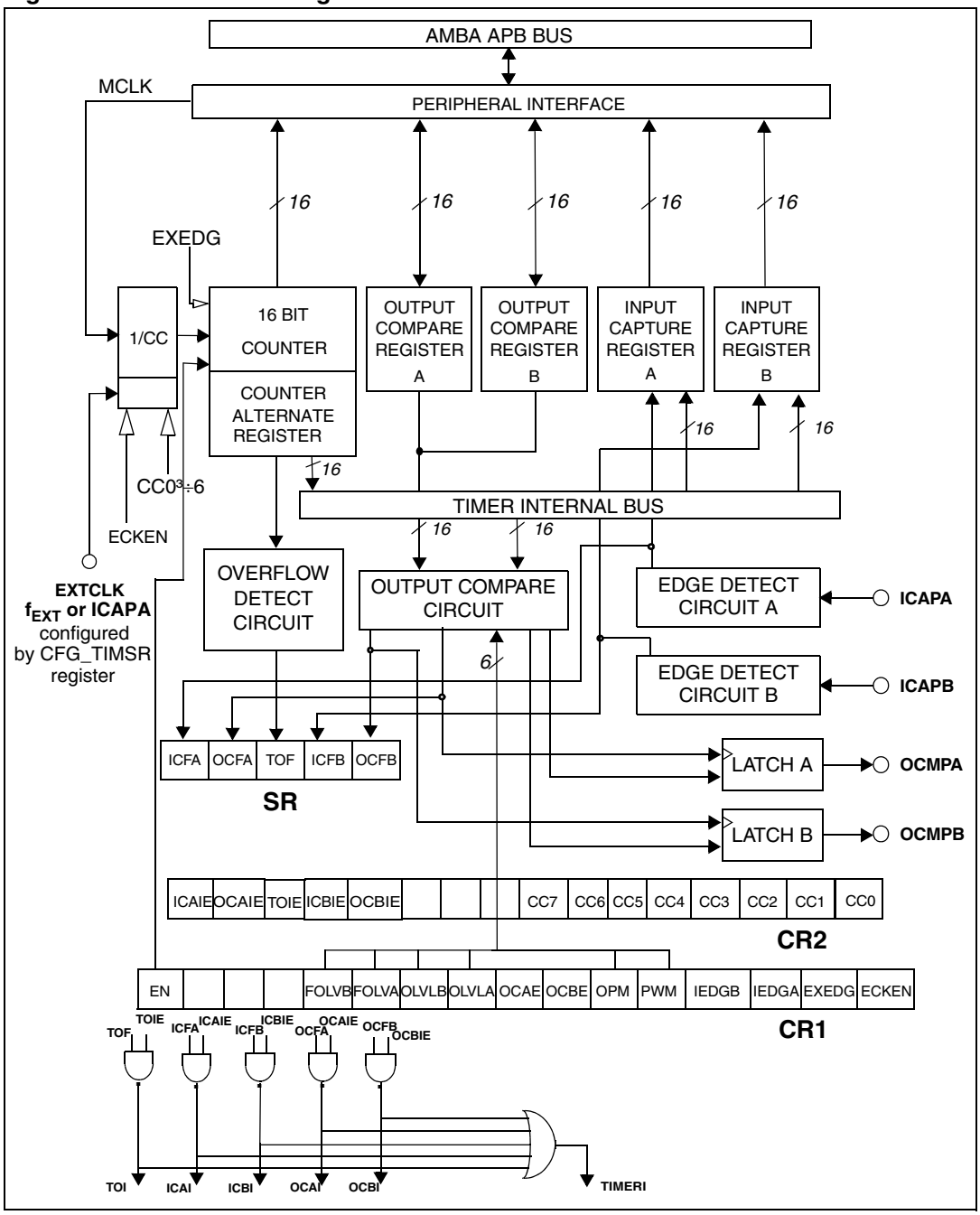
Writing in the Counter Register (CNTR) resets the counter to the FFFCh value.

The timer clock source can be either internal or external selecting ECKEN bit of CR1 register. When ECKEN = 0, the frequency depends on the prescaler division bits (CC7-CC0) of the CR2 register.

An overflow occurs when the counter rolls over from FFFFh to 0000h then the TOF bit of the SR register is set. An interrupt is generated if TOIE bit of the CR2 register is set; if this condition is false, the interrupt remains pending to be issued as soon as it becomes true.

Clearing the overflow interrupt request is done by a write access to the SR register while the TOF bit is set with the data bus 13-bit at ‘0’, while all the other bits shall be written to ‘1’ (the SR register is clear only, so writing a ‘1’ in a bit has no effect: this makes possible to clear a pending bit without risking to clear a new coming interrupt request from another source).

Figure 33. Timer block diagram



14.3.2 External clock

The external clock is selected if ECKEN = 1 in CR1 register.

The source of the external clock is either f_{EXT} or the ICAPAn pin. This is configured by the CFG_TIMSR register (see *Section 4.3.6*). For a description of f_{EXT} refer to *Section 3.6*.

The status of the EXEDG bit determines the type of level transition on the external clock pin EXTCLK that triggers the counter.

The counter is synchronized with the rising edge of MCLK.

At least four rising edges of MCLK must occur between two consecutive active edges of the external clock; thus the external clock frequency must be less than a quarter of the MCLK frequency.

Figure 34. Counter timing diagram, internal clock divided by 2

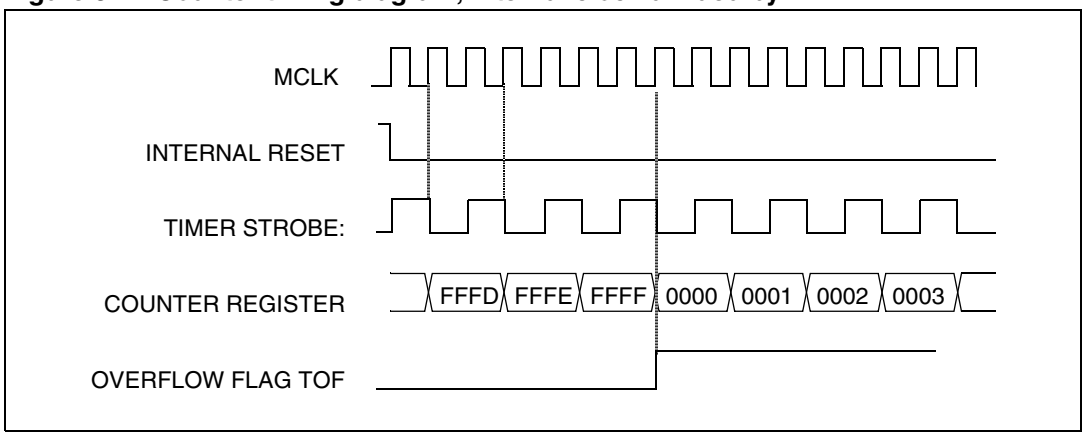


Figure 35. Counter timing diagram, internal clock divided by 4

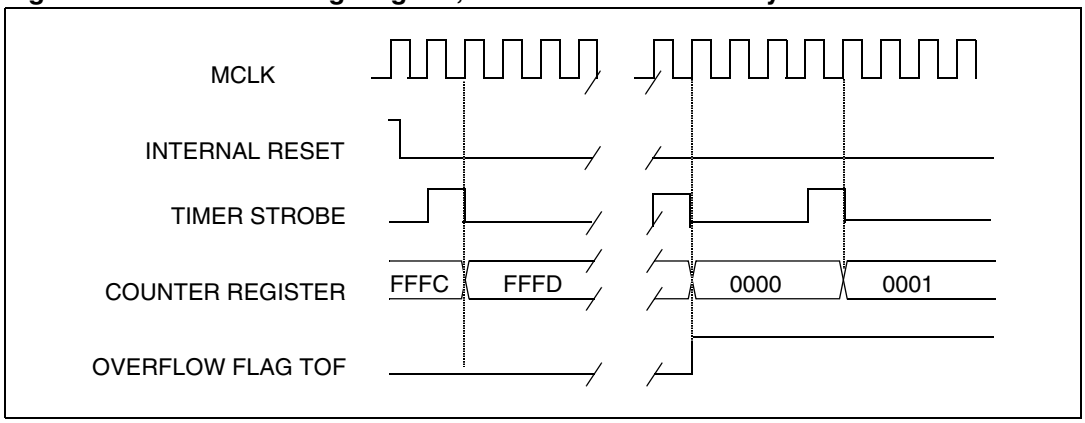
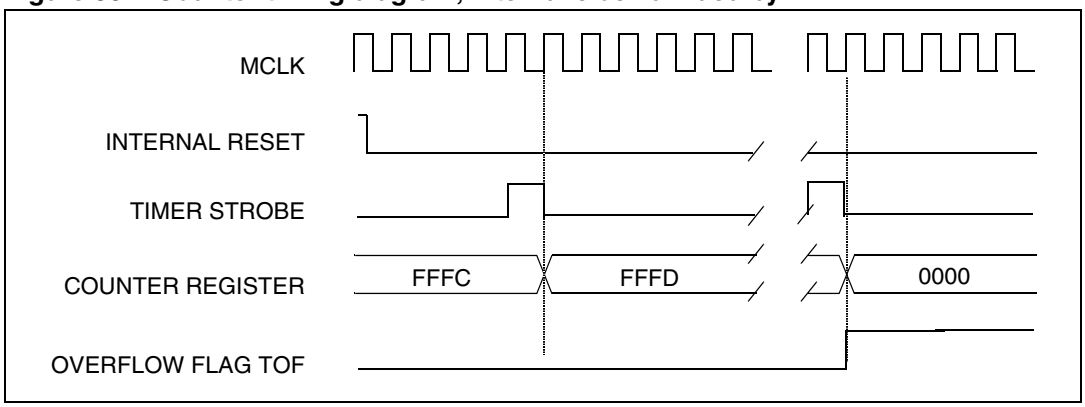


Figure 36. Counter timing diagram, internal clock divided by n



According to particular device implementation, the external clock can be available on a general purpose I/O pin as alternate function, or internally connected to a reference clock independent from the PLL: this allows the Timer to count events independently by the

system clock (which could be prescaled or multiplied according to the different run and low-power modes), generating regular time basis.

14.3.3 Input capture

In this section, the index "*i*", may be A or B.

The two input capture 16-bit registers (ICAR and ICBR) are used to latch the value of the counter after a transition detected by the ICAP*i* pin (see *Figure 37*).

IC*i*R register are read-only registers.

The active transition is software programmable through the IEDG*i* bit of the Control Register (CR1).

Timing resolution is one/two count of the counter: ($f_{MCLK/([CC1..CC0]+1)}$).

14.3.4 Procedure

To use the input capture function select the following in the CR1 and CR2 registers:

- Select the timer clock source (ECKEN).
- Select the timer clock division factor (CC7÷CC0) if internal clock is used.
- Select the edge of the active transition on the ICAPA pin with the IEDGA bit, if ICAPA is active.
- Select the edge of the active transition on the ICAPB pin with the IEDGB bit, if ICAPB is active.
- Set ICAIE (or ICBIE) when ICAPA (or ICAPB) is active, to generate an interrupt after an input capture.

When an input capture occurs:

- ICF*i* bit is set.
- The IC*i*R register contains the value of the counter on the active transition on the ICAP*i* pin (see *Figure 38*).
- A timer interrupt is generated if ICAIE is set (if only ICAPA is active) or ICBIE is set (if only ICAPB is active); otherwise, the interrupt remains pending until concerned enable bits are set.

Clearing the Input Capture interrupt request is done by:

1. A write access to the SR register while the ICF*i* bit is cleared, 15-bit at '0' for ICAPA and 12-bit at '0' for ICAPB.
2. When an active level is detected in the DMA acknowledge signal, the DMA is enabled and the appropriate DMA source is selected.

Figure 37. Input capture block diagram

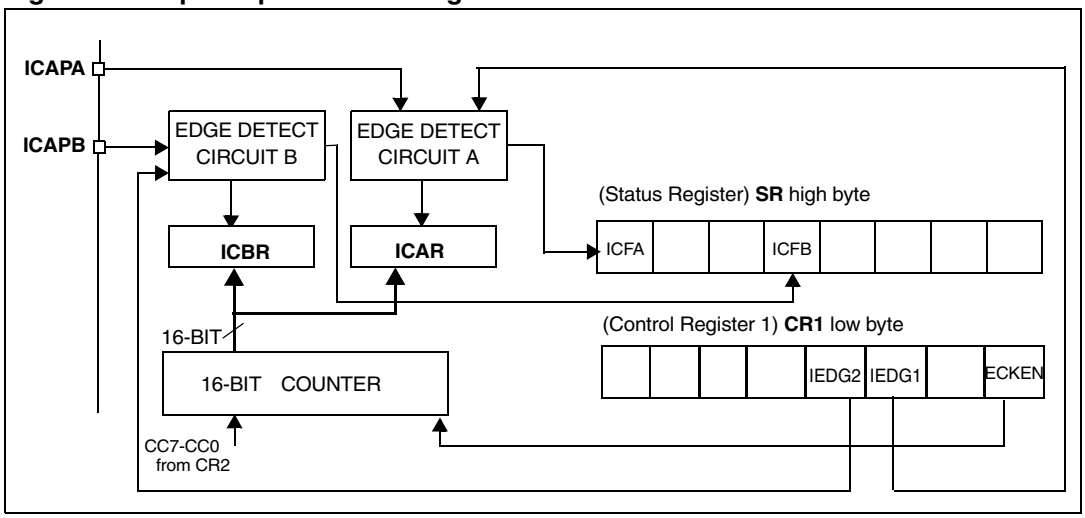
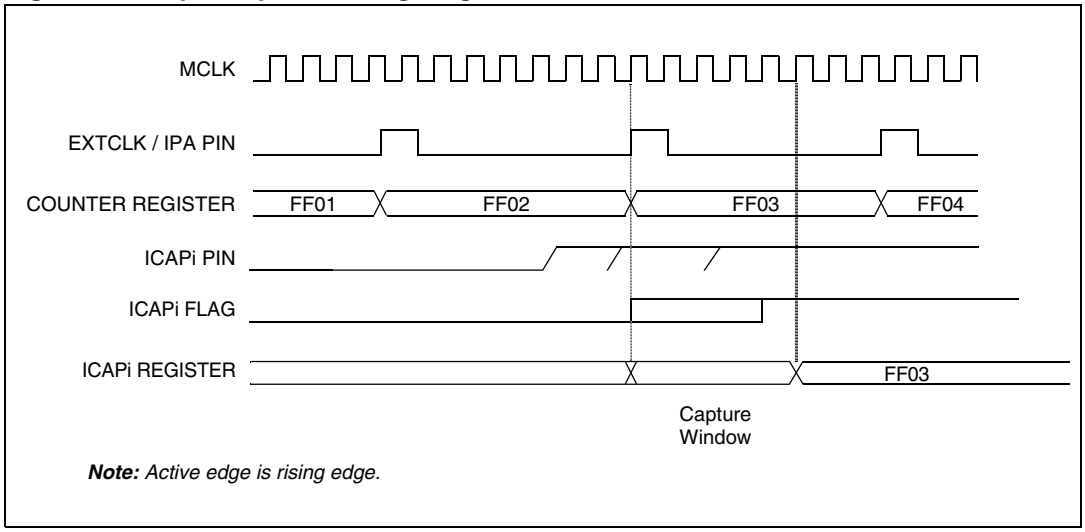


Figure 38. Input capture timing diagram



Note: *Active edge is rising edge.*

14.3.5 Output compare

In this section, the index "*i*", may be A or B.

This function can be used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the Output Compare register and the counter, the output compare function:

- Assigns pins with a programmable value if the OC*i*E bit is set
- Sets a flag in the status register
- Generates an interrupt if enabled

Two 16-bit registers Output Compare Register A (OCAR) and Output Compare Register B (OCBR) contain the value to be compared to the counter each timer clock cycle.

These registers are readable and writable and are not affected by the timer hardware. A reset event changes the OC*i*R value to 8000h.

Timing resolution is one count of the counter: ($f_{MCLK/([CC7..CC0]+1)}$).

14.3.6 Procedure

To use the output compare function, select the following in the CR1/CR2 registers:

- Set the OC*i*E bit if an output is needed then the OCMP*i* pin is dedicated to the output compare *i* function.
- Select the timer clock (ECKGEN) and the prescaler division factor (CC7-CC0).

Select the following in the CR1/CR2 registers:

- Select the OLVL*i* bit to applied to the OCMP*i* pins after the match occurs.
- Set OCAIE (OCBIE) if only compare A (compare B) needs to generate an interrupt.

When match is found:

- OCF*i* bit is set.
- The OCMP*i* pin takes OLVL*i* bit value (OCMP*i* pin latch is forced low during reset and stays low until valid compares change it to OLVL*i* level).
- A timer interrupt is generated if the OCAIE (or OCBIE) bit in CR2 register is set, the OCAR (or OCBR) matches the timer counter (i.e. OCFA or OCFB is set).

Clearing the output compare interrupt request is done by a write access to the SR register while the OCF*i* bit is cleared, 14-bit at '0' for OCAR and 12-bit at '0' for OCBR.

If the OC*i*E bit is not set, the OCMP*i* pin is at '0' and the OLVL*i* bit will not appear when match is found.

The value in the 16-bit OC*i*R register and the OLVL*i* bit should be changed after each successful comparison in order to control an output waveform or establish a new elapsed timeout.

The OC*i*R register value required for a specific timing application can be calculated using the following formula:

$$\Delta \text{ OC}i\text{R} = \frac{\Delta t * f_{MCLK}}{([CC7..CC0]+1)}$$

Where:

Δt = Desired output compare period (in seconds)
f_{MCLK} = Internal clock frequency
CC7-CC0 = Timer clock prescaler

Figure 39. Output compare block diagram

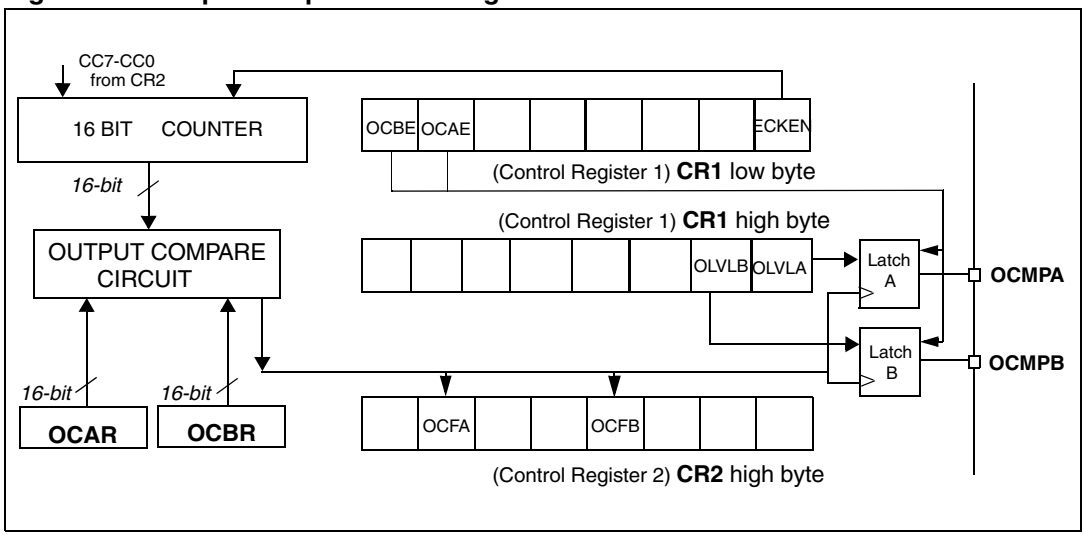
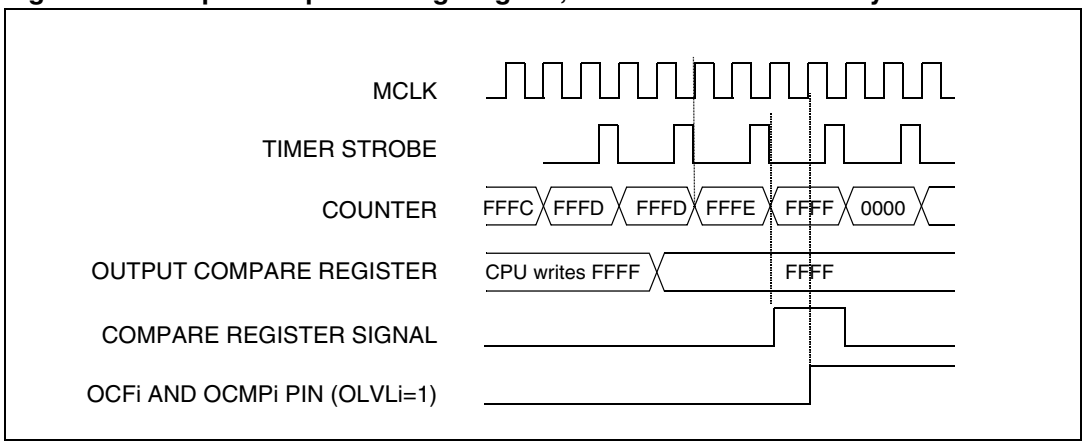


Figure 40. Output compare timing diagram, internal clock divided by 2



14.3.7 Forced compare mode

In this section the index "*i*" may represent A or B.

Bits 11:8 of CR1 register and bits 7:0 of CR2 are used (Refer to *Section 14.6* for detailed Register Description).

When the FOLVA bit is set, the OLVLA bit is copied to the OCMPA pin if PWM and OPM are both cleared. When FOLVB bit is set, the OLVLB bit is copied to the OCMPB pin.

The OLVL*i* bit has to be toggled in order to toggle the OCMP*i* pin when it is enabled (OC*i*E bit=1).

Note: 1 When FOLVi is set, no interrupt request is generated.

2 Nevertheless the OCFi bit can be set if OCiR = Counter, an interrupt can be generated if enabled.

3 Input capture function works in Forced Compare mode.

14.3.8 One pulse mode

One pulse mode enables the generation of a pulse when an external event occurs. This mode is selected via the OPM bit in the CR1 register.

The one pulse mode uses the Input Capture A function (trigger event) and the Output Compare A function.

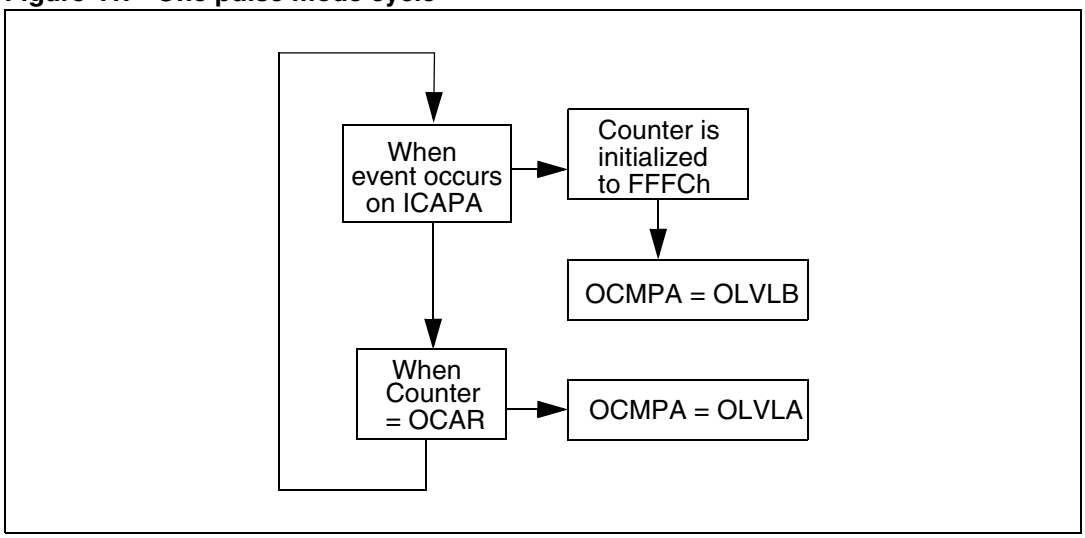
14.3.9 Procedure

To use one pulse mode, select the following in the CR1 register:

- Using the OLVLA bit, select the level to be applied to the OCMPA pin after the pulse.
- Using the OLVLB bit, select the level to be applied to the OCMPA pin during the pulse.
- Select the edge of the active transition on the ICAPA pin with the IEDGA bit.
- Set the OCAE bit, the OCMPA pin is then dedicated to the Output Compare A function.
- Set the OPM bit.
- Select the timer clock (ECKGEN) and the prescaler division factor (CC7-CC0).

Load the OCAR register with the value corresponding to the length of the pulse (see the formula in next *Section 14.3.11*).

Figure 41. One pulse mode cycle



Then, on a valid event on the ICAPA pin, the counter is initialized to FFFCh and OLVLB bit is loaded on the OCMPA pin after four clock period. When the value of the counter is equal to the value of the contents of the OCAR register, the OLVLA bit is output on the OCMPA pin (See *Figure 42*).

Note: 1 The OCFA bit cannot be set by hardware in one pulse mode but the OCFB bit can generate an Output Compare interrupt.

2 The ICFA bit is set when an active edge occurs and can generate an interrupt if the ICAIE bit is set. The ICAR register will have the value FFFCh.

3 When the Pulse Width Modulation (PWM) and One Pulse Mode (OPM) bits are both set with FOLVA= 1, the OPM mode is the only active one, otherwise the PWM mode is the only active one.

4 Forced Compare B mode works in OPM

5 Input Capture B function works in OPM

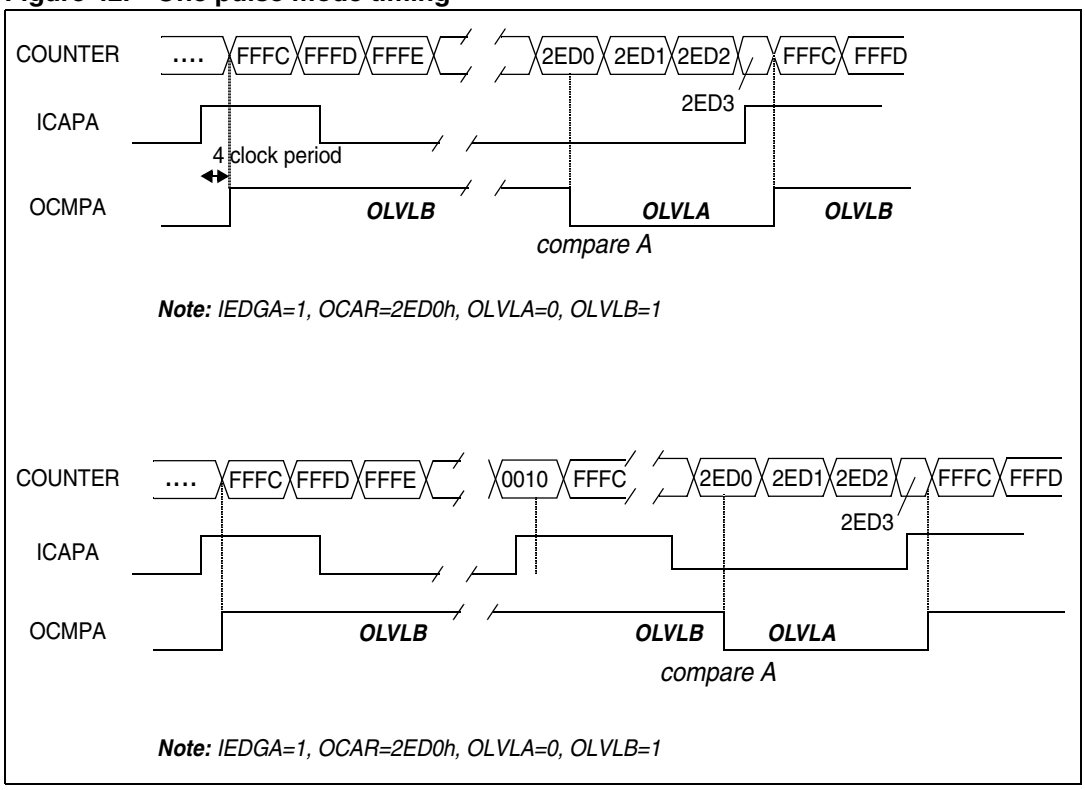
6 When OCAR = FFFBh in OPM, then a pulse of width FFFFh is generated

7 If event occurs on ICAPA again before the Counter reaches the value of OCAR, then the Counter will be reset again and the pulse generated might be longer than expected as in Figure 42.

8 If a write operation is performed on the counter register before the Counter reaches the value of OCAR, then the Counter will be reset again and the pulse generated might be longer than expected.

9 If a write operation is performed on the counter register after the Counter reaches the value of OCAR, then there will have no effect on the waveform.

Figure 42. One pulse mode timing



14.3.10 Pulse width modulation mode

Pulse Width Modulation mode enables the generation of a signal with a frequency and pulse length determined by the value of the OCAR and OCBR registers.

The pulse width modulation mode uses the complete Output Compare A function plus the OCBR register.

14.3.11 Procedure

To use pulse width modulation mode, select the following in the CR1 register:

- Using the OLVLA bit, select the level to be applied to the OCMPA pin after a successful comparison with OCAR register.
- Using the OLVLB bit, select the level to be applied to the OCMPA pin after a successful comparison with OCBR register.
- Set OCAE bit: the OCMPA pin is then dedicated to the output compare A function.
- Set the PWM bit.
- Select the timer clock (ECKGEN) and the prescaler division factor (CC7-CC0).

Load the OCBR register with the value corresponding to the period of the signal.

Load the OCAR register with the value corresponding to the length of the pulse if (OLVLA=0 and OLVLB=1).

If OLVLA=1 and OLVLB=0 the length of the pulse is the difference between the OCBR and OCAR registers.

The OC*i*R register value required for a specific timing application can be calculated using the following formula:

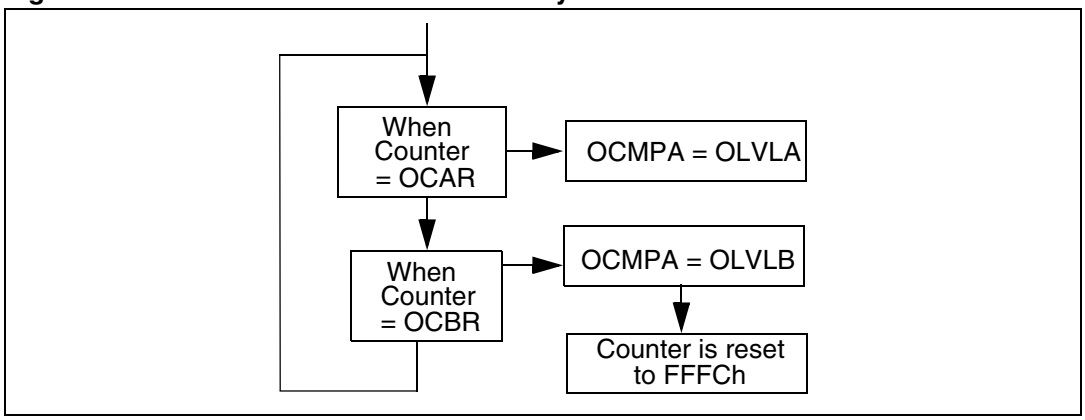
$$\text{OC}i\text{R Value} = \frac{t * f_{MCLK}}{t_{PRESC}} - 5$$

Where:

t = Desired output compare period (seconds)
f_{MCLK} = Internal clock frequency (Hertz)
t_{PRESC} = Timer clock prescaler (1, 2 ... , 256)

The Output Compare B event causes the counter to be initialized to FFFCh (See *Figure 44*).

Figure 43. Pulse width modulation mode cycle



Note: 1 The OCFA bit cannot be set by hardware in PWM mode, but OCFB is set every time counter matches OCBR.

2 The Input Capture function is available in PWM mode.

3 When Counter = OCBR, then OCFB bit will be set. This can generate an interrupt if OCBIE is set. This interrupt will help any application where pulse-width or period needs to be changed interactively.

4 When the Pulse Width Modulation (PWM) and One Pulse Mode (OPM) bits are both set with FOLVA = 0, the PWM mode is the only active one, otherwise the OPM mode is the only active one.

*5 The value loaded in OCBR **must always be greater than** that in OCAR to produce meaningful waveforms. Note that 0000h is considered to be greater than FFFCh or FFFDh or FFFEh or FFFFh.*

6 When OCAR > OCBR, no waveform will be generated.

*7 When OCBR = OCAR, a **square** waveform with 50% duty cycle will be generated as in Figure 44.*

8 When OCBR > OCAR:

$$\text{Period} = t_{APB2} \times (\text{PRESC} + 1) \times (\text{OCBR} - \text{FFFC} + 1)$$

9 When OCBR and OCAR are loaded with FFFCh (the counter reset value) then a square waveform will be generated & the counter will remain stuck at FFFCh. The period will be calculated using the following formula:

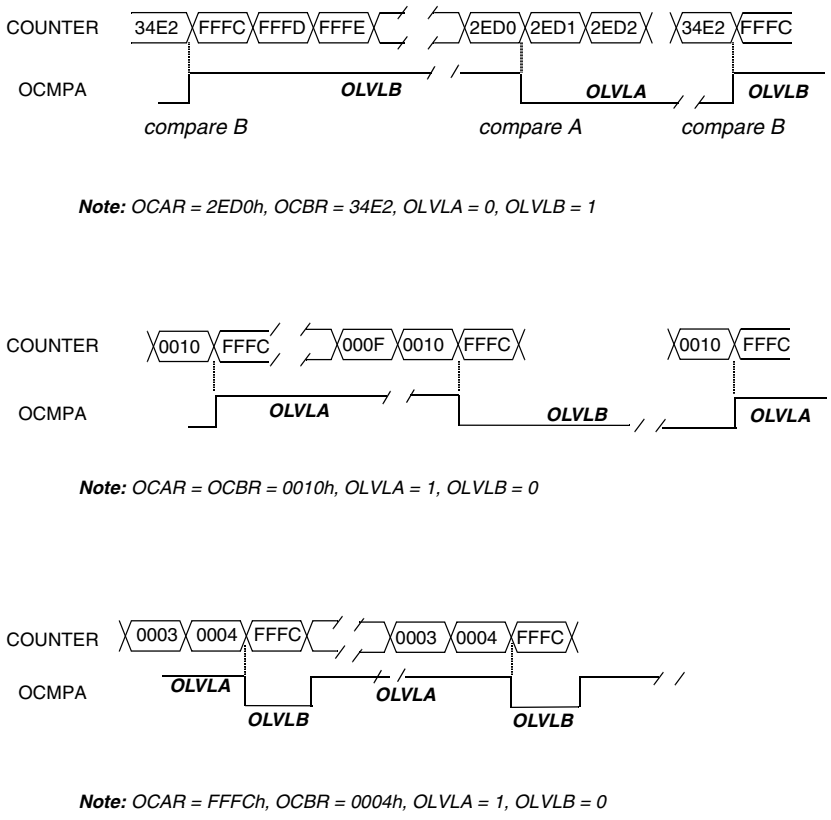
$$\text{Period} = t_{APB2} \times (\text{PRESC} + 1) \times (\text{OCBR} - \text{FFFC} + 1) \times 2$$

10 When OCAR is loaded with FFFCh (the counter reset value) then the waveform will be generated as in Figure 44.

11 When FOLVA bit is set and PWM bit is set, then PWM mode is the active one. But if FOLVB bit is set then the OLVLB bit will appear on OCMPB (when OCBE bit = 1).

12 When a write is performed on CNTR register in PWM mode, then the Counter will be reset and the pulse-width/period of the waveform generated may not be as desired.

Figure 44. Pulse width modulation mode timing



14.3.12 Pulse width modulation input

The PWM Input functionality enables the measurement of the period and the pulse width of an external waveform. The initial edge is programmable.

It uses the two Input Capture registers and the Input signal of the Input Capture A module.

14.3.13 Procedure

The CR2 register must be programmed as needed for Interrupt and DMA generation. To use pulse width modulation mode select the following in the CR1 register:

- set the PWMI bit
- Select the first edge in IEDGA
- Select the second edge IEDGB as the negated of IEDGA
- Program the clock source and prescaler as needed
- Enable the counter setting the EN bit.

To have a coherent measurement the interrupt/DMA should be linked to the Input Capture A Interrupt, reading in ICAR the period value and in ICBR the pulse width.

To obtain the time values:

$$\text{Period} = \frac{ICAR * f_{MCLK}}{t_{PRESC}}$$

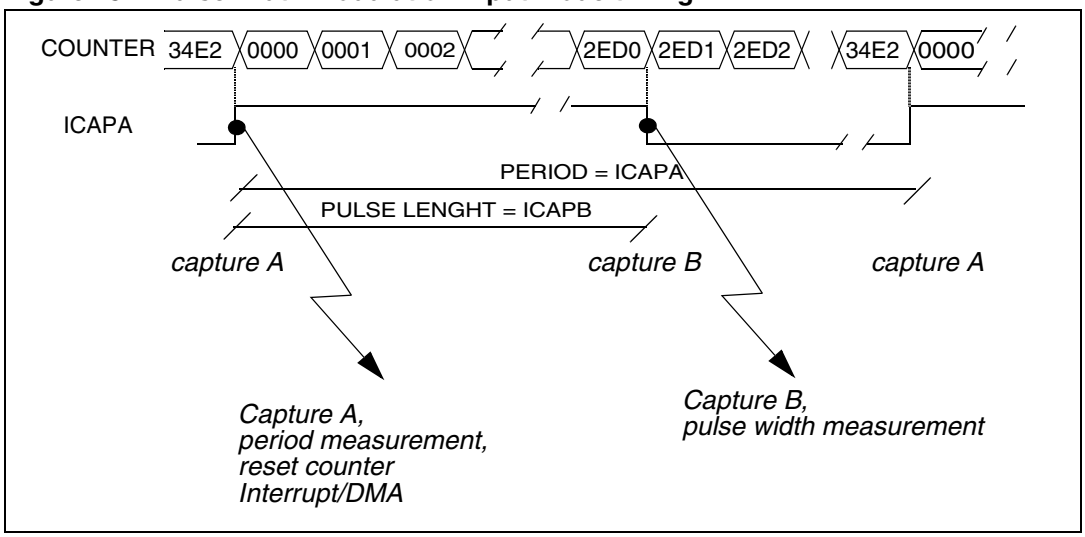
$$\text{Pulse} = \frac{ICBR * f_{MCLK}}{t_{PRESC}}$$

Where:

f_{MCLK} = Internal clock frequency
t_{PRESC} = Timer clock prescaler

The Input Capture A event causes the counter to be initialized to 0000h, allowing a new measure to start. The first Input Capture on ICAPA do not generate the corresponding interrupt/DMA request.

Figure 45. Pulse width modulation input mode timing



14.4 Interrupt management

The three interrupt sources are available both mapped on five different interrupt channels and also being mapped on the same channel.

14.4.1 Use of interrupt channels

To use the interrupt features, for each interrupt channel used, perform the following sequence:

- Set the OC*i*E and/or IC*i*E and/or TOIE bits of CR2 register to enable the peripheral to perform interrupt requests on the desired events

The selection of the five or single interrupt channels is performed by connecting the desired interrupt wire(s) to the interrupt controller when the timer peripheral is instantiated inside a system.

14.5 DMA function

On DMA interface is available on the whole timer module; the source can be selected to be one of ICAPA, OCMPA, ICAPB, OCMPB.

To use the DMA feature:

– Select the DMA source programming the DMAS0/DMAS1 bits in CR1

– Set the DMAIE bit in CR2 register.

14.6 Register description

Each Timer is associated with two control and one status registers, and with six pairs of data registers (16-bit values) relating to the two input captures, the two output compares, the counter. Every register can have only an access by 16 bits, that means is not possible to read or write only a byte.

14.6.1 Input Capture A Register (TIMn_ICAR)

Address Offset: 00h
Reset value: xxxxh

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

This is a 16-bit read only register that contains the counter value transferred by the Input Capture A event.

14.6.2 Input capture B register (TIMn_ICBR)

Address Offset: 04h
Reset value: xxxxh

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

This is a 16-bit read only register that contains the counter value transferred by the Input Capture B event.

14.6.3 Output compare A register (TIMn_OCAR)

Address Offset: 08h
Reset value: 8000h

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

This is a 16-bit register that contains the value to be compared to the CNTR register and signalled on OCMPA output.

14.6.4 Output compare B register (TIMn_OCBR)

Address Offset: 0Ch
Reset value: 8000h

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

This is a 16-bit register that contains the value to be compared to the CNTR register and signalled on OCMPB output.

14.6.5 Counter register (TIMn_CNTR)

Address Offset: 10h
Reset value: FFFCh

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

This is a 16-bit register that contains the counter value. By writing in this register the counter is reset to the FFFCh value.

14.6.6 Control register 1 (TIMn_CR1)

Address Offset: 14h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	PWMI	DMAS 1	DMAS 0	FOLVB	FOLVA	OLVLB	OLVLA	OCBE	OCAE	OPM	PWM	IEDGB	IEDGA	EXED G	ECKEN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 = **EN**: *Timer Count Enable*
0: Timer counter is stopped.
1: Timer counter is enabled.

Bit 14 = **PWMI**: *Pulse Width Modulation Input*
0: PWM Input is not active.
1: PWM Input is active.

Bits 13:12 = **DMAS0-DMAS1**: *DMA source select*
00: ICAPA used as DMA source.
01: OCMPA used as DMA source.
10: ICAPB used as DMA source.
11: OCMPB used as DMA source.

Bit 11 = **FOLVB**: *Forced Output Compare B*
0: No effect.
1: Forces OLVLB to be copied to the OCMPB pin.

Bit 10 = **FOLVA**: *Forced Output Compare A*
0: No effect.
1: Forces OLVLA to be copied to the OCMPA pin.

Bit 9 = **OLVLB**: *Output Level B*
This bit is copied to the OCMPB pin whenever a successful comparison occurs with the OCBR register and OCBE is set in the CR2 register. This value is copied to the OCMPA pin in One Pulse Mode and Pulse Width Modulation mode.

Bit 8= **OLVLA**: *Output Level A*
The OLVLA bit is copied to the OCMPA pin whenever a successful comparison occurs with the OCAR register and the OCAE bit is set in the CR2 register.

Bit 7= **OCBE**: *Output Compare B Enable*
0: Output Compare B function is enabled, but the OCMPB pin is a general I/O.
1: Output Compare B function is enabled, the OCMPB pin is dedicated to the Output Compare B capability of the timer.

Bit 6= **OCAE**: *Output Compare A Enable*
0: Output Compare A function is enabled, but the OCMPA pin is a general I/O.
1: Output Compare A function is enabled, the OCMPA pin is dedicated to the Output Compare A capability of the timer.

Bit 5 = **OPM**: *One Pulse Mode*
0: One Pulse Mode is not active.
1: One Pulse Mode is active, the ICAPA pin can be used to trigger one pulse on the OCMPA pin; the active transition is given by the IEDGA bit. The length of the generated pulse depends on the contents of the OCAR register.

Bit 4 = **PWM**: *Pulse Width Modulation*
0: PWM mode is not active.
1: PWM mode is active, the OCMPA pin outputs a programmable cyclic signal; the length of the pulse depends on the value of OCAR register; the period depends on the value of OCBR register.

Bit 3 = **IEDGB**: *Input Edge B*
This bit determines which type of level transition on the ICAPB pin will trigger the capture.
0: A falling edge triggers the capture.
1: A rising edge triggers the capture.

Bit 2 = **IEDGA**: *Input Edge A*
This bit determines which type of level transition on the ICAPA pin will trigger the capture.
0: A falling edge triggers the capture.
1: A rising edge triggers the capture.

Bit 1 = **EXEDG**: *External Clock Edge*
This bit determines which type of level transition on the external clock pin (or internal signal) EXTCLK will trigger the counter.
0: A falling edge triggers the counter.
1: A rising edge triggers the counter.

Bit 0 = **ECKEN**: *External Clock Enable*
0: Internal clock, divided by prescaler division factor, is used to feed timer clock.
1: External source is used for timer clock.

14.6.7 Control register 2 (TIMn_CR2)

Address Offset: 18h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICAIE	OCAIE	TOE	ICBIE	OCBIE	DMAIE	Reserved		CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0
rw	rw	rw	rw	rw	rw	-		rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 = **ICAIE**: *Input Capture A Interrupt Enable*
0: No interrupt on input capture A.
1: Generate interrupt if ICFA flag is set.

Bit 14 = **OCAIE**: *Output Compare A Interrupt Enable*
0: No interrupt on OCFA set.
1: Generate interrupt if OCFA flag is set.

Bit 13 = **TOIE**: *Timer Overflow Interrupt Enable*
0: Interrupt is inhibited.
1: A timer interrupt is enabled whenever the TOF bit of the SR register is set.

Bit 12= **ICBIE**: *Input Capture B Interrupt Enable*
0: No interrupt on input capture B.
1: Generate interrupt if ICFB flag is set.

Bit 11 = **OCBIE**: *Output Compare B Interrupt Enable*
0: No interrupt on OCFB set.
1: Generate interrupt if OCFB flag is set.

Bit 10 = **DMAIE**: *DMA Enable*
0: No DMA enabled.
1: DMA enabled on the selected source.

Bits 9:8 = Reserved, must be kept at reset value (0).

Bits 7:0 = **CC[7:0]**: *Prescaler division factor*
This 8-bit string is the factor used by the prescaler to divide the internal clock. Timer clock will be equal to f_{PLCK2} / (CC7:CC0 +1).

14.6.8 Status register (TIMn_SR)

Address Offset: 1Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICFA	OCFA	TOF	ICFB	OCFB	reserved										
rc	rc	rc	rc	rc	-										

Bit 15= **ICFA**: *Input Capture Flag A*
0: No input capture (reset value).
1: An input capture has occurred. To clear this bit, write the SR register, with a ‘0’ on the bit 15 (and ‘1’ in all the other bits, just to avoid an unwanted clearing of another pending bit).

Bit 14= **OCFA**: *Output Compare Flag A*
0: No match (reset value).
1: The content of the counter has matched the content of the OCAR register. This bit is not set in the PWM mode even if counter matches OCAR. To clear this bit, write the SR register, with a ‘0’ on the bit 14 (and ‘1’ in all the other bits, just to avoid an unwanted clearing of another pending bit).

Bit 13= **TOF**: *Timer Overflow*
0: No timer overflow (reset value).
1:The counter rolled over from FFFFh to 0000h. To clear this bit, write the SR register, with a ‘0’ on the bit 13 (and ‘1’ in all the other bits, just to avoid an unwanted clearing of another pending bit).

Bit 12= **ICFB**: *Input Capture Flag B*
0: No input capture (reset value).
1: An input capture has occurred.To clear this bit, write the SR register, with a ‘0’ on the bit 12 (and ‘1’ in all the other bits, just to avoid an unwanted clearing of another pending bit).

Bit 11= **OCFB**: *Output Compare Flag B*
0: No match (reset value).
1: The content of the counter has matched the content of the OCBR register. It is set in PWM mode too. To clear this bit, write the SR register, with a ‘0’ on the bit 11 (and ‘1’ in all the other bits, just to avoid an unwanted clearing of another pending bit).

14.7 TIM register map

Table 32. TIM register map

Addr. offset	Register name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	TIMn_ICAR	Input Capture A															
04h	TIMn_ICBR	Input Capture B															
08h	TIMn_OCAR	Output Compare A															
0Ch	TIMn_OCBR	Output Compare B															
10h	TIMn_CNTR	Counter Value															
14h	TIMn_CR1	EN	PWMI	DMAS1	DMAS0	FOLVB	FOLVA	OLVLB	OLVLA	OCBE	OCAE	OPM	PWM	IEDGB	IEDGA	EXEDG	ECKEN
18h	TIMn_CR2	ICAIE	OCAIE	TOE	ICBIE	OCBIE	DMAIE	reserved		CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0
1Ch	TIMn_SR	ICFA	OCFA	TOF	ICFB	OCFB	reserved										

See *Table 2 on page 17* for base address

15 Pulse width modulator (PWM)

15.1 Introduction

The PWM module can generate PWM signals with programmable period and duty cycle.

15.2 Main features

- Full-scale PWM generation
- Period and duty preload registers
- Programmable PWM output polarities
- PWM output enable
- Compare Period interrupt generation
- PWM input clock prescaled
- Interrupt request

The number of PWM modules depends on the device, each PWM offers a completely independent time basis providing a high level of flexibility.

15.3 Functional description

Two prescalers, PRS0 and PRS1, supply the clock to the PWMs. The first prescaler PRS0 divides the f_{MCLK} by (1, 2, 4, 8,..., 128), the second prescaler PRS1 divides the output of the first one by (1, 2, 3,..., 32).

Figure 46. PWM block diagram

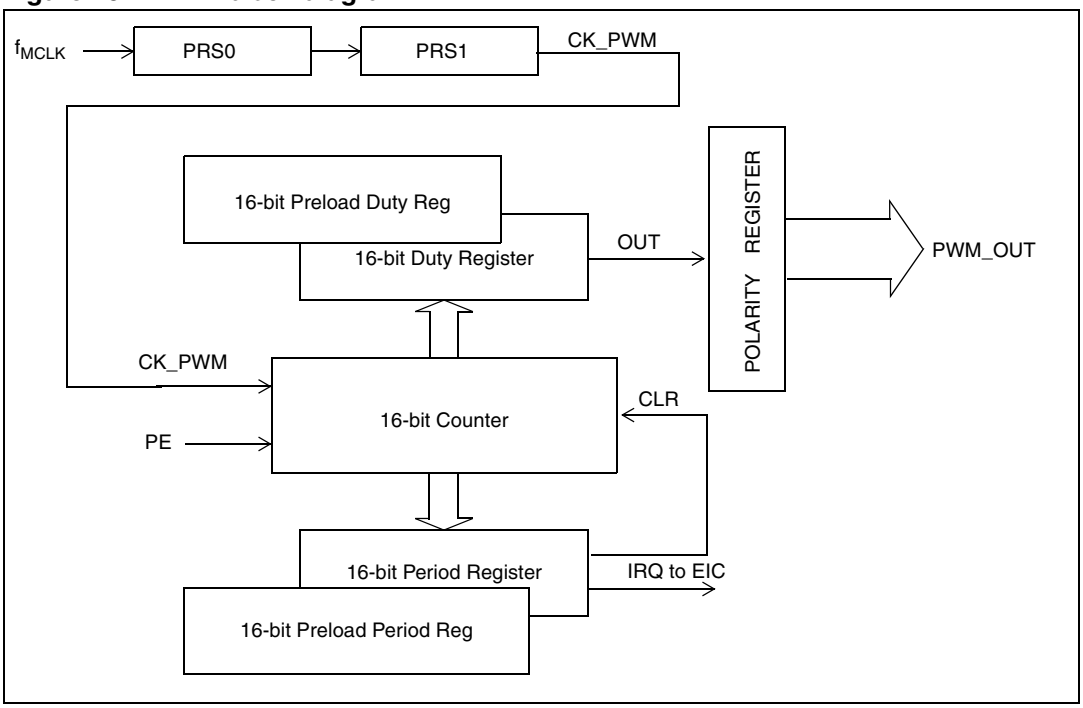


Figure 46 shows the PWM block diagram.

Each PWM channel has two preload registers used as buffers, two effective registers for the period and the duty cycle of PWM output signal and a PWM Counter.

15.3.1 PWM operating mode

After system reset, the PWM Counter is initialized to count '0000' and the Duty Register to '0000', therefore the PWM output OUT is continuously low (see *Figure 47*, wave 4).

When the PWM Enable bit in the PWMn_PEN register is set to '1', the Preload Period and the preload Duty registers are loaded in the effective Period and Duty registers and the PWM counter starts counting. A Compare Period interrupt request is generated.

As long as the PWM counter value is less than the Duty register value, the PWM output OUT is high. When the counter reaches the Duty value, the output goes low.

When PWM counter reaches the Period Register value, the Period and the Duty registers are updated, a Compare Period (CP) interrupt is generated and the PWM Counter restarts counting from '0000' value.

As the PWM Enable bit is cleared, the PWM counter is stopped and reset, the Duty register is cleared and the output OUT becomes low.

The Polarity Level Selection register permits the level inversion of the PWM output.

Note: When both Duty and Period values are loaded one after the other in the preload registers by software may happen that only the first value is updated by hardware because the second value has not been written yet by software. In this case no error condition is generated.

If the value written in the duty register exceed the period register value, the output is continuously high.

15.3.2 Formulas

The clock frequency used to define the resolution of the PWM is computed starting from system clock frequency applying the prescaling factors set into PWMn_PRS0 and PWMn_PRS1 registers.

$$f_{CKPWM} = \frac{f_{MCLK}}{2^{PR0[2 \div 0]} \cdot (PR1[4 \div 0] + 1)}$$

Once the PWM resolution clock is defined, it is possible to program the period of the PWM output wave (T_{PWM}) setting the PWMn_PER register.

$$T_{PWM} = \frac{P[15 \div 0] + 1}{f_{CKPWM}} = \frac{(P[15 \div 0] + 1) \cdot 2^{PR0[2 \div 0]} \cdot (PR1[4 \div 0] + 1)}{f_{MCLK}}$$

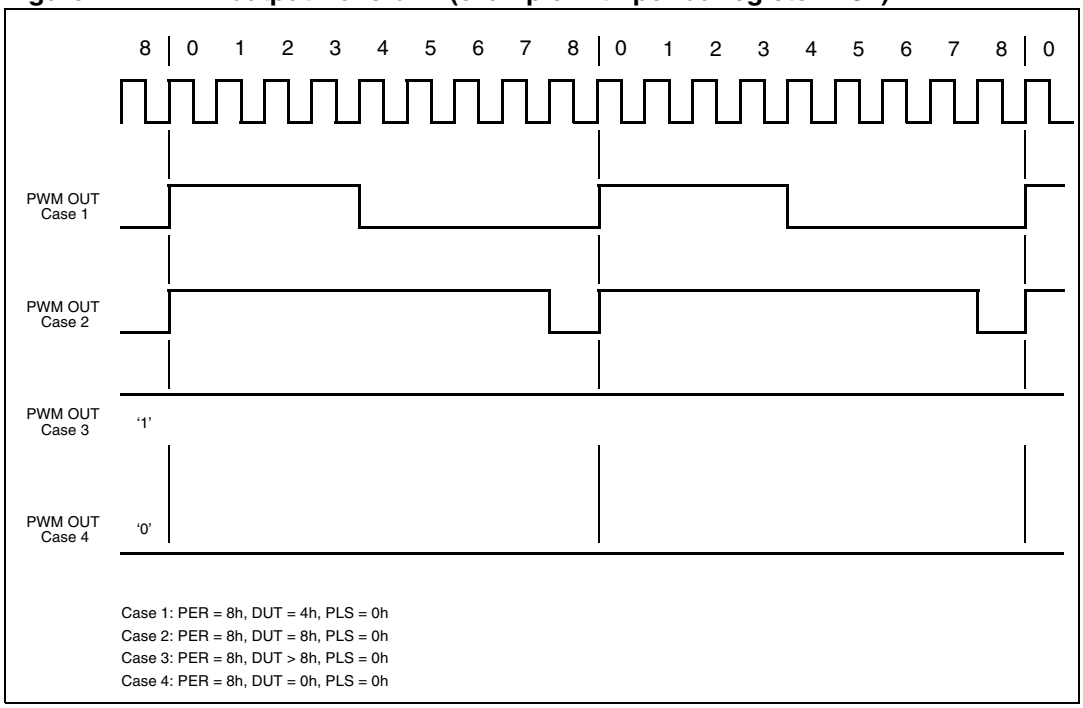
Finally, it is possible to program the duty-cycle value (DC_{PWM}) in percentage of the PWM output wave, setting the PWMn_DUT register.

$$DC_{PWM} = \frac{D[15 \div 0]}{P[15 \div 0] + 1} \cdot 100$$

The minimum value of PWMn_PER register is 0h: this corresponds to a period of 1 clock cycle of CKPWM. Consequently, the Duty-cycle can be either 0% (PWMn_DUT value equal to 0h) or 100% only (PWMn_DUT value greater than PWMn_PER value). Vice versa if the opposite polarity is selected (PWMn_PLS register). When the maximum period setting is programmed (FFFFh), it is not possible to obtain a 100% Duty-cycle (it is not possible to program the PWMn_DUT value greater than the PWMn_PER value): the only way is to set the 0% Duty-cycle and use the polarity to obtain the desired output wave.

Note: The interrupt request is issued at each counter end of count: this means that even though a 100% Duty-cycle output wave is generated by setting PWMn_DUT register value greater than PWMn_PER register value, the PWMn_PER value defines the interrupt request rate.

Figure 47. PWM output waveform (example with period register = 8h)



15.4 Register description

The reserved bits can not be written and they are always read as ‘0’. The registers can not be accessed by byte.

15.4.1 Prescaler 0 register (PWMn_PRS0)

Address Offset: 00h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													PR02	PR01	PR00
-													rw	rw	rw

Bits 15:3 = Reserved, must be kept at reset value (0).

Bits 2:0 = **PR0[2:0]:** *Prescaler 0 value.*
The input clock is divided by $2^{PR0[2:0]}$. Possible values of divider factor: 1, 2, 4, 8,..., 128.
The valid values of PR0[2:0] are in the range from 0 to 7.

15.4.2 Prescaler 1 register (PWMn_PRS1)

Address Offset: 04h
Reset value: 0000h

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

Bits 15:5 = Reserved, must be kept at reset value (0).

Bit 4:0 = **PR1[4:0]:** *Prescaler 1 value.*
The input clock is divided by PR1[4:0] + 1. Possible values of divider factor: 1, 2, 3,..., 32.
The valid values of PR1[4:0] are in the range from 0 to 31.

15.4.3 PWM enable register (PWMn_PEN)

Address Offset: 08h
Reset value: 0000h

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

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **PE:** *PWM enable bit.*
0: PWM output is tied to the reset value.
1: PWM output is ongoing.

15.4.4 PWM output polarity level selection (PWMn_PLS)

Address Offset: 0Ch
Reset value: 0000h

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

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **PL:** *PWM output level polarity bit.*
0: PWM output is not inverted.
1: PWM output is inverted.

15.4.5 PWM compare period interrupt (PWMn_CPI)

Address Offset: 10h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															CP
-															rc

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **CP:** *PWM Compare Period Interrupt bit.*
Every time the PWM counter reaches the PWMn_PER register value, the CP bit is set and an interrupt request is generated if enabled. This bit is set only by hardware and must be cleared by software.

15.4.6 PWM interrupt mask register (PWMn_IM)

Address Offset: 14h
Reset value: 0000h

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

Bits 15:1 = Reserved, must be kept at reset value (0).

Bit 0 = **IM:** *PWM interrupt mask bit.*
0: Compare Period PWM interrupt to EIC is disabled.
1: Compare Period PWM interrupt to EIC is enabled.

15.4.7 PWM output duty register (PWMn_DUT)

Address Offset: 1Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **D[15:0]:** *Duty of PWM output.*
PWM output duty cycle value. It is defined as the number of periods of the PWM clock (CK_PWM) during which the output signal is high (or low if polarity is inverted).
The valid values are in the range from 0 to FFFFh.

15.4.8 PWM output period register (PWMn_PER)

Address Offset: 20h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **P[15:0]:** *Period of the PWM counter.*
PWM output period value. It is defined as the number of PWM clock periods (CKPWM) + 1 (example: when P[15:0]=3, the output PWM period is equal to 4 CKPWM periods).
The valid values are in the range from 0h to FFFFh. Of course, if 0h is programmed, the output is always low if the PWMn_DUT value is 0h as well, on the contrary it is always high if the PWMn_PER value. If inverted polarity is selected (through the PWMn_PLS register) the high and low levels are exchanged

15.5 PWM register map

Table 33. PWM register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	PWMn_PRS0	Reserved													PR02	PR01	PR00
04h	PWMn_PRS1	Reserved											PR14	PR13	PR12	PR11	PR10
08h	PWMn_PEN	Reserved															PE
0Ch	PWMn_PLS	Reserved															PL
10h	PWMn_CPI	Reserved															CP
14h	PWMn_IM	Reserved															IM
18h	-	Reserved															
1Ch	PWMn_DUT	DUT[15:0]															
20h	PWMn_PER	PER[15:0]															

See *Table 2 on page 17* for base address

16 Controller area network (CAN)

16.1 Introduction

The C_CAN consists of the CAN Core, Message RAM, Message Handler, Control Registers and Module Interface (Refer to *Figure 48*).

The CAN Core performs communication according to the CAN protocol version 2.0 part A and B. The bit rate can be programmed to values up to 1 MBit/s. For the connection to the physical layer, additional 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. These functions include 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 through the module interface. These registers are used to control/configure the CAN Core and the Message Handler and to access the Message RAM.

16.2 Main features

- Supports CAN protocol version 2.0 part A and B
- Bit rates 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 Re-transmission mode for Time Triggered CAN applications
- Programmable loop-back mode for self-test operation
- Two 16-bit module interfaces to the APB bus

16.3 Block diagram

The C_CAN interfaces with the AMBA APB bus. *Figure 48* shows the block diagram of the C_CAN.

CAN core

CAN Protocol Controller and Rx/Tx Shift Register.

Message RAM

Stores Message Objects and Identifier Masks.

Registers

All registers used to control and to configure the C_CAN.

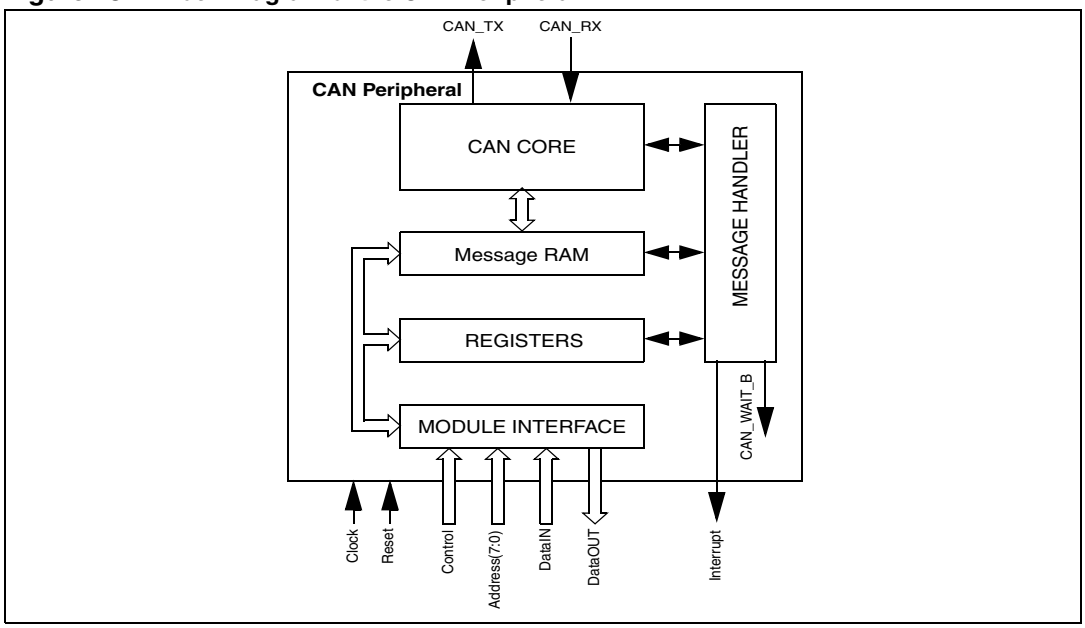
Message handler

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.

Module interface

The module interface provides the interface between the APB 16-bit bus and the C_CAN registers.

Figure 48. Block Diagram of the CAN Peripheral



16.4 Functional description

16.4.1 Software initialization

The software initialization is started by setting the Init bit in the CAN Control Register, either by a software or a hardware reset, or by going to Bus_Off state.

While the Init bit is set, all message transfers to and from the CAN bus are stopped and the status of the CAN_TX output pin is recessive (HIGH). The Error Management Logic (EML) counters are unchanged. Setting the Init bit does not change any configuration register.

To initialize the CAN Controller, software has to set up the Bit Timing Register and each Message Object. If a Message Object is not required, the corresponding MsgVal bit should be cleared. Otherwise, the entire Message Object has to be initialized.

Access to the Bit Timing Register and to the Baud Rate Prescaler (BRP) Extension Register for configuring bit timing is enabled when the Init and Configuration Change Enable *(*CCE) bits in the CAN Control Register are both set.

Resetting the Init bit (by CPU only) finishes the software initialization. Later, the Bit Stream Processor (BSP) (see *Section 16.7.10: Configuring the bit timing on page 217*) 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 start 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, software has to start by resetting the corresponding MsgVal bit. When the configuration is completed, MsgVal is set again.

16.4.2 CAN message transfer

Once the C_CAN is initialized and Init bit is cleared, the C_CAN Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored in 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 are stored in 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.

Software can read or write each message any time through the Interface Registers and the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the application software. If a permanent Message Object (arbitration and control bits are set during configuration) exists for the message, only the data bytes are updated and the 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. Message objects 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.

16.4.3 Disabled automatic re-transmission mode

In accordance with 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 have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. This means that, by default, automatic retransmission is enabled. It can be disabled to enable the C_CAN to work within a Time Triggered CAN (TTCAN, see ISO11898-1) environment.

Disabled Automatic Retransmission mode is enabled by setting the Disable Automatic Retransmission (DAR) bit in the CAN Control Register. 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 cleared, while bit NewDat remains set.
- When the transmission completed successfully, bit NewDat is cleared.
- When a transmission fails (lost arbitration or error), bit NewDat remains set.

To restart the transmission, the CPU should set the bit TxRqst again.

16.4.4 Test mode

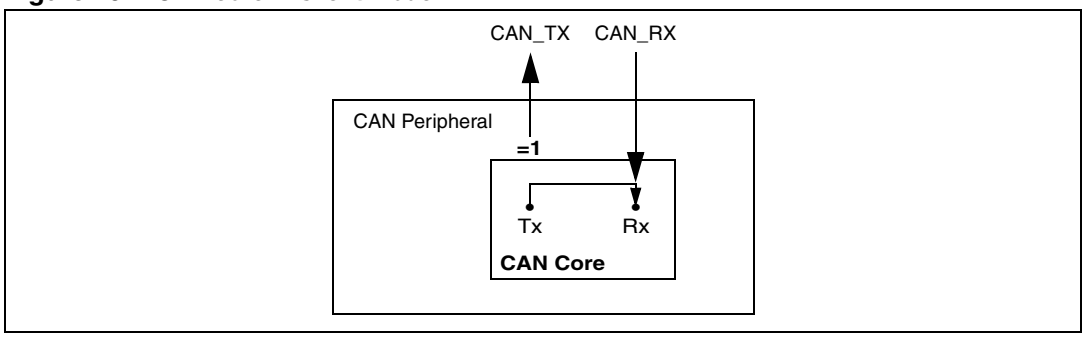
Test Mode is entered by setting the Test bit in the CAN Control Register. In Test Mode, bits Tx1, Tx0, LBack, Silent and Basic in the Test Register are writeable. Bit Rx monitors the state of the CAN_RX pin and therefore is only readable. All Test Register functions are disabled when the Test bit is cleared.

Silent mode

The CAN Core can be set in Silent Mode by programming the Silent bit in the Test Register 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, Error Frames), 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 analyse the traffic on a CAN bus without affecting it by the transmission of *dominant* bits. *Figure 49* shows the connection of signals CAN_TX and CAN_RX to the CAN Core in Silent Mode.

Figure 49. CAN core in silent mode

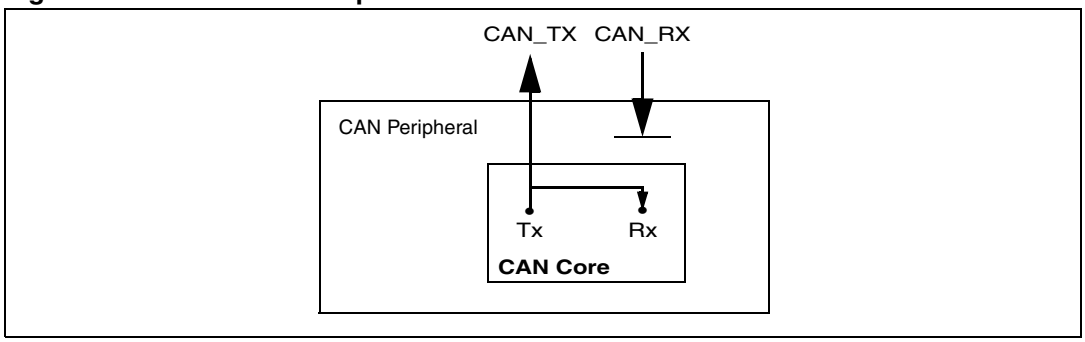


In ISO 11898-1, Silent Mode is called Bus Monitoring Mode.

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 in a Receive Buffer (if they pass acceptance filtering). *Figure 50* shows the connection of signals, CAN_TX and CAN_RX, to the CAN Core in Loop Back Mode.

Figure 50. CAN core in loop back mode

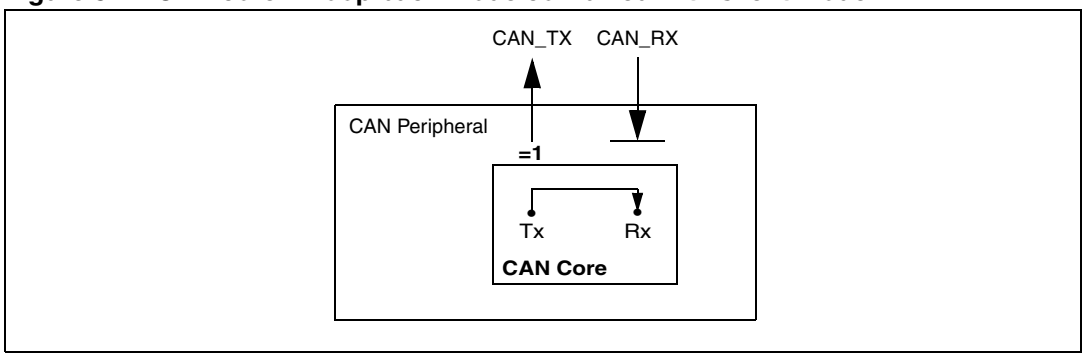


This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode, the CAN Core performs an internal feedback from its Tx output to its Rx input. The actual value of the CAN_RX input pin is disregarded by the CAN Core. The transmitted messages can be monitored on the CAN_TX pin.

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”, which means that C_CAN can be tested without affecting a running CAN system connected to the CAN_TX and CAN_RX pins. In this mode, the CAN_RX pin is disconnected from the CAN Core and the CAN_TX pin is held recessive. *Figure 51* shows the connection of signals CAN_TX and CAN_RX to the CAN Core in case of the combination of Loop Back Mode with Silent Mode.

Figure 51. CAN core in loop back mode combined with silent mode



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 runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers are 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 been 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 a 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 are stored in the IF2 Registers.

In Basic Mode, the evaluation of all Message Object related control and status bits and the control bits of the IFn Command Mask Registers are turned off. The message number of the Command request registers is not evaluated. The NewDat and MsgLst bits in the IF2 Message Control Register retain their function, DLC3-0 indicate the received DLC, and the other control bits are read as '0'.

Software control of CAN_TX pin

Four output functions are available for the CAN transmit pin, CAN_TX. In addition to its default function (serial data output), the CAN transmit pin can drive the CAN Sample Point signal to monitor CAN_Core's bit timing and it can drive constant dominant or recessive values. The latter two functions, combined with the readable CAN receive pin CAN_RX, can be used to check the physical layer of the CAN bus.

The output mode for the CAN_TX pin is selected by programming the Tx1 and Tx0 bits of the CAN Test Register.

The three test functions of the CAN_TX pin interfere with all CAN protocol functions. CAN_TX 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.

16.5 Register description

The C_CAN allocates an address space of 256 bytes. The registers are organized as 16-bit registers.

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.

In this section, the following abbreviations are used:

Table 34. List of abbreviations

read/write (rw)	The software can read and write to these bits.
read-only (r)	The software can only read these bits.
write-only (w)	The software should only write to these bits.

The CAN registers are listed in *Table 35*.

Table 35. CAN registers

Register Name	Address Offset	Reset Value
CAN Control Register (CAN_CR)	00h	0001h
Status Register (CAN_SR)	04h	0000h
Error Counter (CAN_ERR)	08h	0000h
Bit Timing Register (CAN_BTR)	0Ch	2301h
Test Register (CAN_TESTR)	14h	0000 0000 R000 0000 b **Note:** R=current value of the RX pin
BRP Extension Register (CAN_BRPR)	18h	0000h
IFn Command Request Registers (CAN_IFn_CRR)	20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)	0001h
IFn Command Mask Registers (CAN_IFn_CMR)	24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)	0000h
IFn Mask 1 Register (CAN_IFn_M1R)	28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)	FFFFh
IFn Mask 2 Register (CAN_IFn_M2R)	2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)	FFFFh
IFn Message Arbitration 1 Register (CAN_IFn_A1R)	30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)	0000h
IFn Message Arbitration 2 Register (CAN_IFn_A2R)	34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)	0000h
IFn Message Control Registers (CAN_IFn_MCR)	38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)	0000h
IFn Data A/B Registers (CAN_IFn_DAnR and CAN_IFn_DBnR)	3Ch (CAN_IF1_DA1R), 40h (CAN_IF1_DA2R), 44h (CAN_IF1_DB1R), 48h (CAN_IF1_DB2R), 9Ch (CAN_IF2_DA1R), A0h (CAN_IF2_DA2R), A4h (CAN_IF2_DB1R), A8h (CAN_IF2_DB2R)	0000h
Interrupt Identifier Register (CAN_IDR)	10h	0000h
Transmission Request Registers 1 & 2 (CAN_TxRnR)	100h (CAN_TxR1R), 104h (CAN_TxR2R)	0000h
New Data Registers 1 & 2 (CAN_NDnR)	120h (CAN_ND1R), 124h (CAN_ND2R)	0000h
Interrupt Pending Registers 1 & 2 (CAN_IPnR)	140h (CAN_IP1R), 144h (CAN_IP2R)	0000h
Message Valid Registers 1 & 2 (CAN_MVnR)	160h (CAN_MV1R), 164h (CAN_MV2R)	0000h

16.5.1 CAN interface reset state

After the hardware reset, the C_CAN registers hold the reset values givenin *Table 35* and the register descriptions below.

Additionally the *busoff* state is reset and the output CAN_TX 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 the Init bit to '0'.

The data stored in the Message RAM is not affected by a hardware reset. After powering on, the contents of the Message RAM are undefined.

16.5.2 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 (CAN_CR)

Address Offset: 00h
Reset value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Test	CCE	DAR	res	EIE	SIE	IE	Init
-	-	-	-	-	-	-	-	rw	rw	rw	-	rw	rw	rw	rw

Bits 15:8	Reserved, , forced by hardware to 0.
Bit 7	**Test:** *Test Mode Enable* 0: Normal Operation. 1: Test Mode.
Bit 6	**CCE:** *Configuration Change Enable* 0: No write access to the Bit Timing Register. 1: Write access to the Bit Timing Register allowed (while bit Init=1).
Bit 5	**DAR:** *Disable Automatic Re-transmission* 0: Automatic Retransmission of disturbed messages enabled. 1: Automatic Retransmission disabled.
Bit 4	Reserved, forced by hardware to 0.
Bit 3	**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.
Bit 2	**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 1	**IE:** *Module Interrupt Enable* 0: Disabled. 1: Enabled.
Bit 0	**Init:** *Initialization* 0: Normal Operation. 1: Initialization is started.

*Note: The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting the Init bit. If the device goes in the busoff state, it will set Init of its own accord, stopping all bus activities. Once 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 resetting 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 (CAN_SR)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
-	-	-	-	-	-	-	-	r	r	r	rw	rw	rw	rw	rw

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**BOff:** *Busoff Status* 0: The CAN module is not in busoff state. 1: The CAN module is in busoff state.
Bit 6	**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 EML has reached the error warning limit of 96.
Bit 5	**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.
Bit 4	**RxOk:** *Received a Message Successfully* 0: No message has been successfully received since this bit was last reset by the CPU. This bit is never reset by the CAN Core. 1: A message has been successfully received since this bit was last reset by the CPU (independent of the result of acceptance filtering).

Bit 3	**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.
Bits 2:0	**LEC[2:0]:** *Last Error Code (Type of the last error to occur on the CAN bus)* 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 '0' when a message has been transferred (reception or transmission) without error. The unused code '7' may be written by the CPU to check for updates. *Table 36* describes the error codes.

Table 36. Error codes

Error Code	Meaning
0	No Error
1	Stuff Error: More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
2	Form Error: A fixed format part of a received frame has the wrong format.
3	AckError: The message this CAN Core transmitted was not acknowledged by another node.
4	Bit1Error: During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.
5	Bit0Error: During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), though 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 proceedings of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continuously disturbed).
6	CRCError: The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.
7	Unused: When the LEC shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC.

Status interrupts

A Status Interrupt is generated by bits BOff and EWarn (Error Interrupt) or by RxOk, TxOk, and LEC (Status Change Interrupt) assuming 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 (CAN_ERR)

Address Offset: 08h
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	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 15	**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.
Bits 14:8	**REC[6:0]:** *Receive Error Counter* Actual state of the Receive Error Counter. Values between 0 and 127.
Bits 7:0	**TEC[7:0]:** *Transmit Error Counter* Actual state of the Transmit Error Counter. Values between 0 and 255.

Bit timing register (CAN_BTR)

Address Offset: 0Ch

Reset value: 2301h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res	TSeg2			TSeg1				SJW		BRP					
-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15	Reserved, forced by hardware to 0.
Bits 14:12	**TSeg2:** *Time segment after sample point* 0x0-0x7: Valid values for TSeg2 are [0 … 7]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.
Bits 11:8	**TSeg1:** *Time segment before the sample point minus Sync_Seg* 0x01-0x0F: valid values for TSeg1 are [1 … 15]. The actual interpretation by the hardware of this value is such that one more than the value programmed is used.
Bits 7:6	**SJW:** (*Re)Synchronization Jump Width* 0x0-0x3: Valid programmed values are [0 … 3]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.
Bits 5:0	**BRP:** *Baud Rate Prescaler* 0x01-0x3F: The value by which the oscillator 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 [0 … 63]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Note: With a module clock APB_CLK of 8 MHz, the reset value of 0x2301 configures the C_CAN for a bit rate of 500 kBit/s. The registers are only writable if bits CCE and Init in the CAN Control Register are set.

Test register (CAN_TESTR)

Address Offset: 14h
Reset value: 0000 0000 R000 0000 b (R:current value of RX pin)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								Rx	Tx[1:0]		LBack	Silent	Basic	Res	
-	-	-	-	-	-	-	-	r	rw	rw	rw	rw	rw	-	-

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**Rx:** *Current value of* CAN_RX Pin 0: The CAN bus is dominant (CAN_RX = ‘0’). 1: The CAN bus is recessive (CAN_RX = ‘1’).
Bit 6:5	**Tx[1:0]:** CAN_TX pin control 00: Reset value, CAN_TX is controlled by the CAN Core 01: Sample Point can be monitored at CAN_TX pin 10: CAN_TX pin drives a dominant (‘0’) value. 11: CAN_TX pin drives a recessive (‘1’) value.
Bit 4	**LBack:** *Loop Back Mode* 0: Loop Back Mode is disabled. 1: Loop Back Mode is enabled.
Bit 3	**Silent:** *Silent Mode* 0: Normal operation. 1: The module is in Silent Mode.
Bit 2	**Basic:** *Basic Mode* 0: Basic Mode disabled. 1: IF1 Registers used as Tx Buffer, IF2 Registers used as Rx Buffer.
Bits 1:0	Reserved, forced by hardware to 0.

Write access to the Test Register is enabled by setting the Test bit in the CAN Control Register. The different test functions may be combined, but Tx1-0 ≠ “00” disturbs message transfer.

BRP extension register (CAN_BRPR)

Address Offset: 18h
Reset value: 0000h

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

Bits 15:4	Reserved, forced by hardware to 0.
Bits 3:0	**BRPE:** *Baud Rate Prescaler Extension* 0x00-0x0F: By programming BRPE, 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.

16.5.3 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 conflict between the 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 on page 201*) or parts of the Message Object may be transferred between the Message RAM and the IF*n* Message Buffer registers (see *IFn message buffer registers on page 199*) in one single transfer.

The function of the two interface register sets is identical except for the Basic test mode. They can be used the way 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 37: IF1 and IF2 message interface register set on page 195* provides 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 37. IF1 and IF2 message interface register set

Address	IF1 Register Set	Address	IF2 Register Set
CAN Base + 0x20	IF1 Command Request	CAN Base + 0x80	IF2 Command Request
CAN Base + 0x24	IF1 Command Mask	CAN Base + 0x84	IF2 Command Mask
CAN Base + 0x28	IF1 Mask 1	CAN Base + 0x88	IF2 Mask 1
CAN Base + 0x2C	IF1 Mask 2	CAN Base + 0x8C	IF2 Mask 2
CAN Base + 0x30	IF1 Arbitration 1	CAN Base + 0x90	IF2 Arbitration 1
CAN Base + 0x34	IF1 Arbitration 2	CAN Base + 0x94	IF2 Arbitration 2
CAN Base + 0x38	IF1 Message Control	CAN Base + 0x98	IF2 Message Control
CAN Base + 0x3C	IF1 Data A 1	CAN Base + 0x9C	IF2 Data A 1
CAN Base + 0x40	IF1 Data A 2	CAN Base + 0xA0	IF2 Data A 2
CAN Base + 0x44	IF1 Data B 1	CAN Base + 0xA4	IF2 Data B 1
CAN Base + 0x48	IF1 Data B 2	CAN Base + 0xA8	IF2 Data B 2

IF*n* command request registers (CAN_IF*n*_CRR)

Address offset: 20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)
Reset Value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Busy	Reserved									Message Number					
r	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw

A message transfer is started as soon as the application software has written the message number to the Command Request Register. With this write operation, the Busy bit is automatically set to notify the CPU that a transfer is in progress. After a waiting time of 3 to 6 APB_CLK periods, the transfer between the Interface Register and the Message RAM is completed. The Busy bit is cleared.

Bit 15	**Busy:** *Busy Flag* 0: Read/write action has finished. 1: Writing to the IF*n* Command Request Register is in progress. This bit can only be read by the software.
Bits 14:6	Reserved, forced by hardware to 0.
Bits 5:0	**Message Number:** 0x01-0x20: Valid Message Number, the Message Object in the Message RAM is selected for data transfer. 0x00: Not a valid Message Number, interpreted as 0x20. 0x21-0x3F: Not a valid Message Number, interpreted as *0x01-0x1F*.

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.

IF*n* command mask registers (CAN_IF*n*_CMR)

Address offset: 24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)
Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/NewDat	Data A	Data B
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

The control bits of the IF*n* Command Mask Register specify the transfer direction and select which of the IF*n* Message Buffer Registers are source or target of the data transfer.

Bits 15:8	Reserved, forced by hardware to 0.
Bit 7	**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.

Bits 6:0	These bits of IFn Command Mask Register have different functions depending on the transfer direction: **Direction = Write** Bit 6 = Mask Access Mask Bits 0: Mask bits unchanged. 1: transfer Identifier Mask + MDir + MXtd to Message Object. Bit 5 = Arb Access Arbitration Bits 0: Arbitration bits unchanged. 1: Transfer Identifier + Dir + Xtd + MsgVal to Message Object. Bit 4 = Control Access Control Bits 0: Control Bits unchanged. 1: Transfer Control Bits to Message Object. Bit 3 = ClrIntPnd Clear Interrupt Pending Bit When writing to a Message Object, this bit is ignored. Bit 2 = TxRqst/NewDat Access Transmission Request Bit 0: TxRqst bit unchanged. 1: Set TxRqst bit. If a transmission is requested by programming bit TxRqst/NewDat in the IFn Command Mask Register, bit TxRqst in the IFn Message Control Register will be ignored. Bit 1 = Data A Access Data Bytes 3:0 0: Data Bytes 3:0 unchanged. 1: Transfer Data Bytes 3:0 to Message Object. Bit 0 = Data B Access Data Bytes 7:4 0: Data Bytes 7:4 unchanged. 1: Transfer Data Bytes 7:4 to Message Object. **Direction = Read** Bit 6 = Mask: *Access Mask Bits* 0: Mask bits unchanged. 1: Transfer Identifier Mask + MDir + MXtd to IF*n* Message Buffer Register. Bit 5 = Arb: *Access Arbitration Bits* 0: Arbitration bits unchanged. 1: Transfer Identifier + Dir + Xtd + MsgVal to IF*n* Message Buffer Register. Bit 4 = Control: *Access Control Bits* 0: Control Bits unchanged. 1: Transfer Control Bits to IF*n* Message Buffer Register. Bit 3 = ClrIntPnd: *Clear Interrupt Pending Bit* 0: IntPnd bit remains unchanged. 1: Clear IntPnd bit in the Message Object. Bit 2 = 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 IFn Message Control Register always reflect the status before resetting these bits. Bit 1 = Data A Access Data Bytes 3:0 0: Data Bytes 3:0 unchanged. 1: Transfer Data Bytes 3:0 to IF*n* Message Buffer Register. Bit 0 = Data B Access Data Bytes 7:4 0: Data Bytes 7:4 unchanged. 1: Transfer Data Bytes 7:4 to IF*n* Message Buffer Register.

IF*n* 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 201*.

IF*n* mask 1 register (CAN_IF*n*_M1R)

Address offset: 28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)
Reset Value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Msk[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The function of the Msk bits is described in *Message object in the message memory on page 201*.

IF*n* mask 2 register (CAN_IF*n*_M2R)

Address offset: 2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)
Reset Value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MXtd	MDir	Res	Msk[28:16]												
rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The function of the Message Objects bits is described in the *Message object in the message memory on page 201*.

IF*n* message arbitration 1 register (CAN_IF*n*_A1R)

Address offset: 30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)
Reset Value: 0000h

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

The function of the Message Objects bits is described in the *Message object in the message memory on page 201*.

IF*n* message arbitration 2 register (CAN_IF*n*_A2R)

Address offset: 34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)
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	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The function of the Message Objects bits is described in the *Message object in the message memory on page 201*.

IF*n* message control registers (CAN_IF*n*_MCR)

Address offset: 38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)
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	Reserved			DLC[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	rw	rw	rw	rw

The function of the Message Objects bits is described in the *Message object in the message memory on page 201*.

IF*n* data A/B registers (CAN_IF*n*_DA*n*R and CAN_IF*n*_DB*n*R)

The data bytes of CAN messages are stored in the IF*n* Message Buffer Registers in the following order:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Message Data A1 (address 0x3C)	Data(1)								Data(0)							
IF1 Message Data A2 (address 0x40)	Data(3)								Data(2)							
IF1 Message Data B1 (address 0x44)	Data(5)								Data(4)							
IF1 Message Data B2 (address 0x48)	Data(7)								Data(6)							
IF2 Message Data A1 (address 0x9C)	Data(1)								Data(0)							
IF2 Message Data A2 (address 0xA0)	Data(3)								Data(2)							
IF2 Message Data B1 (address 0xA4)	Data(5)								Data(4)							
IF2 Message Data B2 (address 0xA8)	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.

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 through the IF*n* Interface Registers.

Table 38 provides an overview of the structures of a Message Object

Table 38. Structure of a message object in the message memory

Message Object											
UMask	Msk 28-0	MXtd	MDir	EoB	NewDat		MsgLst	RxIE	TxIE	Int Pnd	RmtEn	TxRqst
MsgVal	ID28-0	Xtd	Dir	DLC 3-0	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

The Arbitration Registers ID28-0, Xtd, and Dir are used to define the identifier and type of outgoing messages and are used (together with the mask registers Msk28-0, MXtd, and MDir) for acceptance filtering of incoming messages. A received message is stored in the valid Message Object with matching identifier and direction set to receive (Data Frame) or transmit (Remote Frame). Extended frames can be stored only in Message Objects with Xtd set, standard frames in Message Objects with Xtd clear. If a received message (Data Frame or Remote Frame) matches 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 212*.

MsgVal	Message Valid 1: The Message Object is configured and should be considered by the Message Handler. 0: The Message Object is ignored by the Message Handler. Note: The application software 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 Id28-0, the control bits Xtd, Dir, or the Data Length Code DLC3-0 are modified, or if the Messages Object is no longer required.
UMask	Use Acceptance Mask 1: Use Mask (Msk28-0, MXtd, and MDir) for acceptance filtering. 0: Mask ignored. Note: 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.
ID28-0	Message Identifier ID28 - ID0, 29-bit Identifier ("Extended Frame") ID28 - ID18, 11-bit Identifier ("Standard Frame")
Msk28-0	Identifier Mask 1: The corresponding identifier bit is used for acceptance filtering. 0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.

Xtd	Extended Identifier 1: The 29-bit (“extended”) Identifier will be used for this Message Object. 0: The 11-bit (“standard”) Identifier will be used for this Message Object.
MXtd	Mask Extended Identifier 1: The extended identifier bit (IDE) is used for acceptance filtering. 0: The extended identifier bit (IDE) has no effect on the 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 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). 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.
MDir	Mask Message Direction 1: The message direction bit (Dir) is used for acceptance filtering. 0: The message direction bit (Dir) has no effect on the acceptance filtering.
EoB	End of Buffer 1: Single Message Object or last Message Object of a FIFO Buffer. 0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer. Note: This bit is used to concatenate two or 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 16.7.7: Configuring a FIFO buffer*.
NewDat	New Data 1: The Message Handler or the application software has written new data into the data portion of this Message Object. 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 application software.
MsgLst	Message Lost (only valid for Message Objects with direction = *receive*) 1: The Message Handler stored a new message into this object when NewDat was still set, the CPU has lost a message. 0: No message lost since last time this bit was reset by the CPU.
RxIE	Receive Interrupt Enable 1: IntPnd will be set after a successful reception of a frame. 0: IntPnd will be left unchanged after a successful reception of a frame.
TxIE	Transmit Interrupt Enable 1: IntPnd will be set after a successful transmission of a frame. 0: IntPnd will be left unchanged after the successful transmission of a frame.

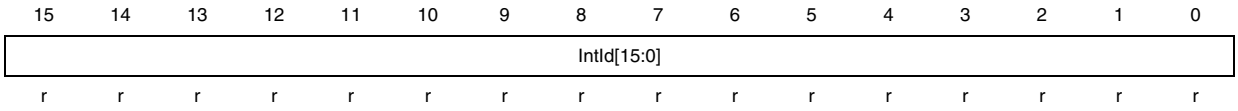
IntPnd	Interrupt Pending 1: This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority. 0: This message object is not the source of an interrupt.
RmtEn	Remote Enable 1: At the reception of a Remote Frame, TxRqst is set. 0: At the reception of a Remote Frame, TxRqst is left unchanged.
TxRqst	Transmit Request 1: The transmission of this Message Object is requested and is not yet done. 0: This Message Object is not waiting for transmission.
DLC3-0	Data Length Code *0-8*: Data Frame has 0-8 data bytes. *9-15* : 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:** 1st data byte of a CAN Data Frame **Data 1**: 2nd data byte of a CAN Data Frame **Data 2**: 3rd data byte of a CAN Data Frame **Data 3**: 4th data byte of a CAN Data Frame **Data 4**: 5th data byte of a CAN Data Frame **Data 5**: 6th data byte of a CAN Data Frame **Data 6**: 7th data byte of a CAN Data Frame **Data 7:** 8th data byte of a CAN Data Frame Note: The Data 0 Byte is the first data byte shifted into the shift register of the CAN Core during a reception while the Data 7 byte 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 unspecified values.

16.5.4 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 identifier register (CAN_IDR)

Address Offset: 10h
Reset value: 0000h



Bits 15:0	**IntId[15:0]:** Interrupt Identifier (*Table 39* indicates the source of the interrupt) 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 application software has cleared it. If IntId is different from 0x0000 and IE is set, the IRQ interrupt signal to the EIC is active. The interrupt 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.

Table 39. Source of interrupts

Interrupt Identifier	Cause of the Interrupt
0x0000	No Interrupt is Pending
0x0001-0x0020	Number of Message Object which caused the interrupt.
0x0021-0x7FFF	unused
0x8000	Status Interrupt
0x8001-0xFFFF	unused

Transmission request registers 1 & 2 (CAN_TxR*n*R)

Address Offset: 100h (CAN_TxR1R), 104h (CAN_TxR2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TxRqst[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TxRqst[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

These registers hold the TxRqst bits of the 32 Message Objects. By reading the TxRqst bits, the CPU can check which Message Object in a Transmission Request is pending. The TxRqst bit of a specific Message Object can be set/reset by the application software through the IFn Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

Bits 31:16	**TxRqst[32:17]:** *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 bits are read only.
Bits 15:0	**TxRqst1[6:1]:** *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 bits are read only.

New data registers 1 & 2 (CAN_ND*n*R)

Address Offset: 120h (CAN_ND1R), 124h (CAN_ND2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NewDat[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NewDat[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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 through the IFn Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

Bits 31:16	**NewDat[32:17]:** *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 the last time this flag was cleared by the application software. 1: The Message Handler or the application software has written new data into the data portion of this Message Object.
Bits 15:0	**NewDat[16:1]:** *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 the last time this flag was cleared by the application software. 1: The Message Handler or the application software has written new data into the data portion of this Message Object.

Interrupt pending registers 1 & 2 (CAN_IP*n*R)

Address Offset: 140h (CAN_IP1R), 144h (CAN_IP2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IntPnd[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IntPnd[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

These registers contain the IntPnd bits of the 32 Message Objects. By reading 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 application software through the IFn 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

Bits 31:16	**IntPnd[32:17]:** *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.
Bits 15:0	**IntPnd[16:1]:** *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.

Message valid registers 1 & 2 (CAN_MV*n*R)

Address Offset: 160h (CAN_MV1R), 164h (CAN_MV2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MsgVal[32:17]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal[16:1]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

These registers hold the MsgVal bits of the 32 Message Objects. By reading the MsgVal bits, the application software can check which Message Object is valid. The MsgVal bit of a specific Message Object can be set/reset by the application software via the IFn Message Interface Registers.

Bits 31:16	**MsgVal[32:17]:** *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.
Bits 15:0	**MsgVal[16:1]:** *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.

16.6 CAN register map

Table 40. CAN register map

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	CAN_CR	Reserved								Test	CCE	DAR	res	EIE	SIE	IE	Init
04h	CAN_SR	Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
08h	CAN_ERR	RP	REC6-0							TEC7-0							
0Ch	CAN_BTR	res	TSeg2			TSeg1				SJW		BRP					
10h	CAN_IDR	IntId15-8								IntId7-0							
14h	CAN_TESTR	Reserved								Rx	Tx1	Tx0	LBack	Silent	Basic	Reserved	
18h	CAN_BRPR	Reserved												BRPE			
20h	CAN_IF1_CRR	Busy	Reserved									Message Number					
24h	CAN_IF1_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
28h	CAN_IF1_M1R	Msk15-0															
2Ch	CAN_IF1_M2R	MXtd	MDir	res	Msk28-16												
30h	CAN_IF1_A1R	ID15-0															
34h	CAN_IF1_A2R	Msg Val	Xtd	Dir	ID28-16												
38h	CAN_IF1_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
3Ch	CAN_IF1_DA1R	Data(1)								Data(0)							
40h	CAN_IF1_DA2R	Data(3)								Data(2)							
44h	CAN_IF1_DB1R	Data(5)								Data(4)							
48h	CAN_IF1_DB2R	Data(7)								Data(6)							
80h	CAN_IF2_CRR	Busy	Reserved									Message Number					
84h	CAN_IF2_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
88h	CAN_IF2_M1R	Msk15-0															
8Ch	CAN_IF2_M2R	MXtd	MDir	res	Msk28-16												
90h	CAN_IF2_A1R	ID15-0															

Table 40. CAN register map (continued)

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
94h	CAN_IF2_A2R	Msg Val	Xtd	Dir	ID28-16												
98h	CAN_IF2_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
9Ch	CAN_IF2_DA1R	Data(1)								Data(0)							
A0h	CAN_IF2_DA2R	Data(3)								Data(2)							
A4h	CAN_IF2_DB1R	Data(5)								Data(4)							
A8h	CAN_IF2_DB2R	Data(7)								Data(6)							
100h	CAN_TxR1R	TxRqst16-1															
104h	CAN_TxR2R	TxRqst32-17															
120h	CAN_ND1R	NewDat16-1															
124h	CAN_ND2R	NewDat32-17															
140h	CAN_IP1R	IntPnd16-1															
144h	CAN_IP2R	IntPnd32-17															
160h	CAN_MV1R	MsgVal16-1															
164h	CAN_MV2R	MsgVal32-17															

Note: Reserved bits are read as 0' except for IFn Mask 2 Register where they are read as '1'.

Refer to *Table 2 on page 17* for the register base addresses.

16.7 CAN communications

16.7.1 Managing message objects

The configuration of the Message Objects in the Message RAM (with the exception of the bits MsgVal, NewDat, IntPnd, and TxRqst) will not be affected by resetting the chip. All the Message Objects must be initialized by the application software or they must be "not valid" (MsgVal = '0') and the bit timing must be configured before the application software clears the Init bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data fields of one of the two interface registers to the desired values. By writing to the corresponding IFn Command Request Register, the IFn 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 state machine of the Message Handler control the internal data flow of the C_CAN. Received messages that pass the acceptance filtering are stored in

the Message RAM, messages with pending transmission request are loaded into the CAN_Core's Shift Register and are transmitted through the CAN bus.

The application software reads received messages and updates messages to be transmitted through the IF*n* Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

16.7.2 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 IF*n* Registers.

The Message Handler FSM controls the following functions:

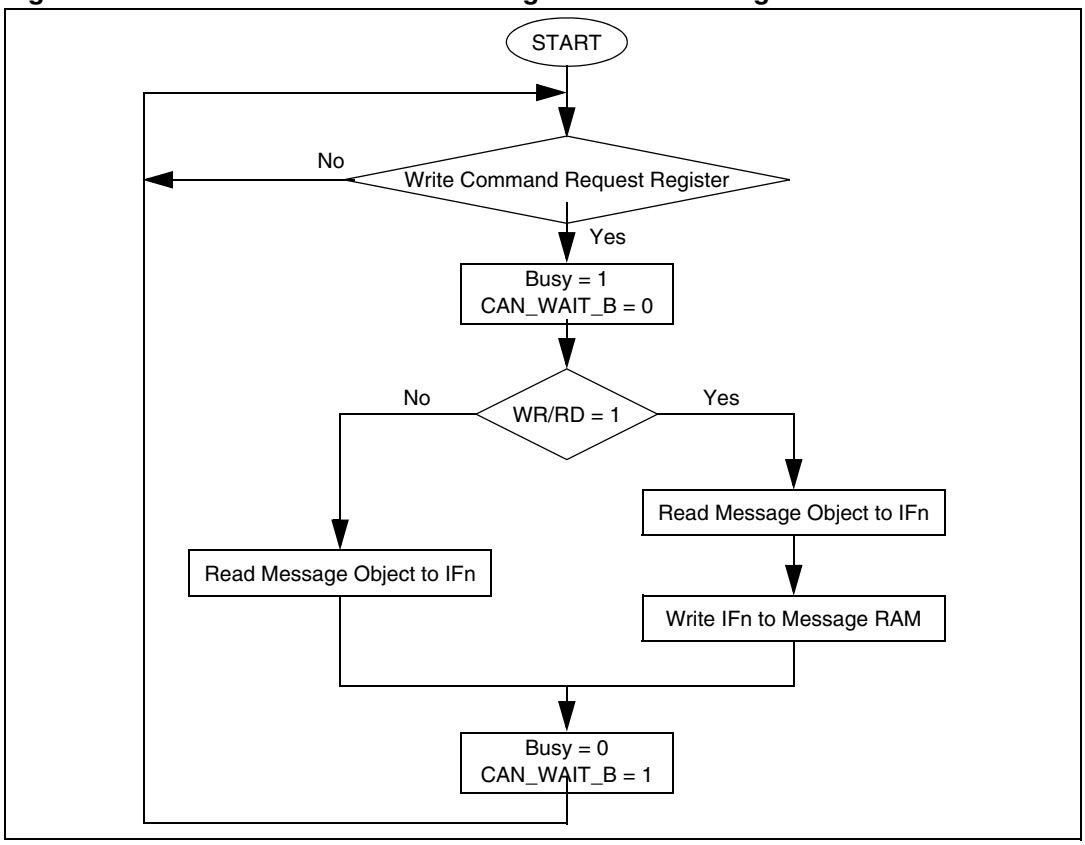
- Data Transfer from IF*n* Registers to the Message RAM
- Data Transfer from Message RAM to the IF*n* Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of 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 IF*n* Registers and Message RAM, the Message Handler sets the Busy bit in the respective Command Request Register (CAN_IFn_CRR). After the transfer has completed, the Busy bit is again cleared (see *Figure 52*).

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 IF*n* Registers to the Message RAM requires a read-modify-write cycle. First, those parts of the Message Object that are not to be changed are read from the Message RAM and then the complete contents of the Message Buffer Registers are written into the Message Object.

Figure 52. Data transfer between IF*n* Registers and Message RAM



After a partial write of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will set the actual contents of the selected Message Object.

After a partial read of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be left unchanged.

Message transmission

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFn 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 NewDat bit of the Message Object is reset.

After a successful transmission and also if no new data was written to the Message Object (NewDat = ‘0’) since the start of the transmission, the TxRqst bit of the Message Control register (CAN_IFn_MCR) will be reset. If TxIE bit of the Message Control register (CAN_IFn_MCR) is set, IntPnd bit of the Interrupt Identifier register (CAN_IDR) 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. Meanwhile, if 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. The arbitration and mask fields (including MsgVal, UMask, NewDat, and EoB) of Message Object 1 are then 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 scan 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 in the corresponding Message Object. This is done 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 application software should reset NewDat bit when the Message Object has been read. If at the time of 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 in the Message Object of 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 due to the priority of the corresponding Message Object.

16.7.3 Configuring a transmit object

Table 41 shows how a Transmit Object should be initialized.

Table 41. Initialization of a Transmit Object

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 Register values (ID28-0 and Xtd bit) are provided 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 ID28 - ID18. The ID17 - ID0 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 Register values (DLC3-0, Data0-7) are provided by the application, TxRqst and RmtEn may not be set before the data is valid.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask=’1’) to allow groups of Remote Frames with similar identifiers to set the TxRqst bit. The Dir bit should not be masked.

16.7.4 Updating a transmit object

The CPU may update the data bytes of a Transmit Object any time through the IFn 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 IFn Data A Register or IFn Data B Register have to be valid before the contents of that register are transferred to the Message Object. Either the CPU has to write all four bytes into the IFn Data Register or the Message Object is transferred to the IFn Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x0087 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 *Message transmission on page 211*.

When NewDat is set together with TxRqst, NewDat will be reset as soon as the new transmission has started.

16.7.5 Configuring a receive object

Table 42 shows how a Receive Object should be initialized.

Table 42. Initialization of a Receive Object

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 values (ID28-0 and Xtd bit) are provided 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 ID28 - ID18. Then ID17 - ID0 can be disregarded. When a Data Frame with an 11-bit Identifier is received, ID17 - ID0 will be set 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 (DLC3-0) is provided 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 unspecified values.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask=’1’) to allow groups of Data Frames with similar identifiers to be accepted. The Dir bit should not be masked in typical applications.

16.7.6 Handling received messages

The CPU may read a received message any time via the IF*n* Interface registers. The data consistency is guaranteed by the Message Handler state machine.

Typically, the CPU will write first 0x007F to the Command Mask Register and then the number of the Message Object to the Command Request Register. This 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 shows which of the matching messages have been received.

The actual value of NewDat shows whether a new message has been received since the last time this Message Object was read. The actual value of MsgLst shows whether more than one message has been received since the 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.

16.7.7 Configuring 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 16.7.5: Configuring a receive object on page 214*.

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 End of the Block.

16.7.8 Receiving messages with FIFO buffers

Received messages with identifiers matching to a FIFO Buffer are stored in a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored in 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 access by the Message Handler until the application software 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.

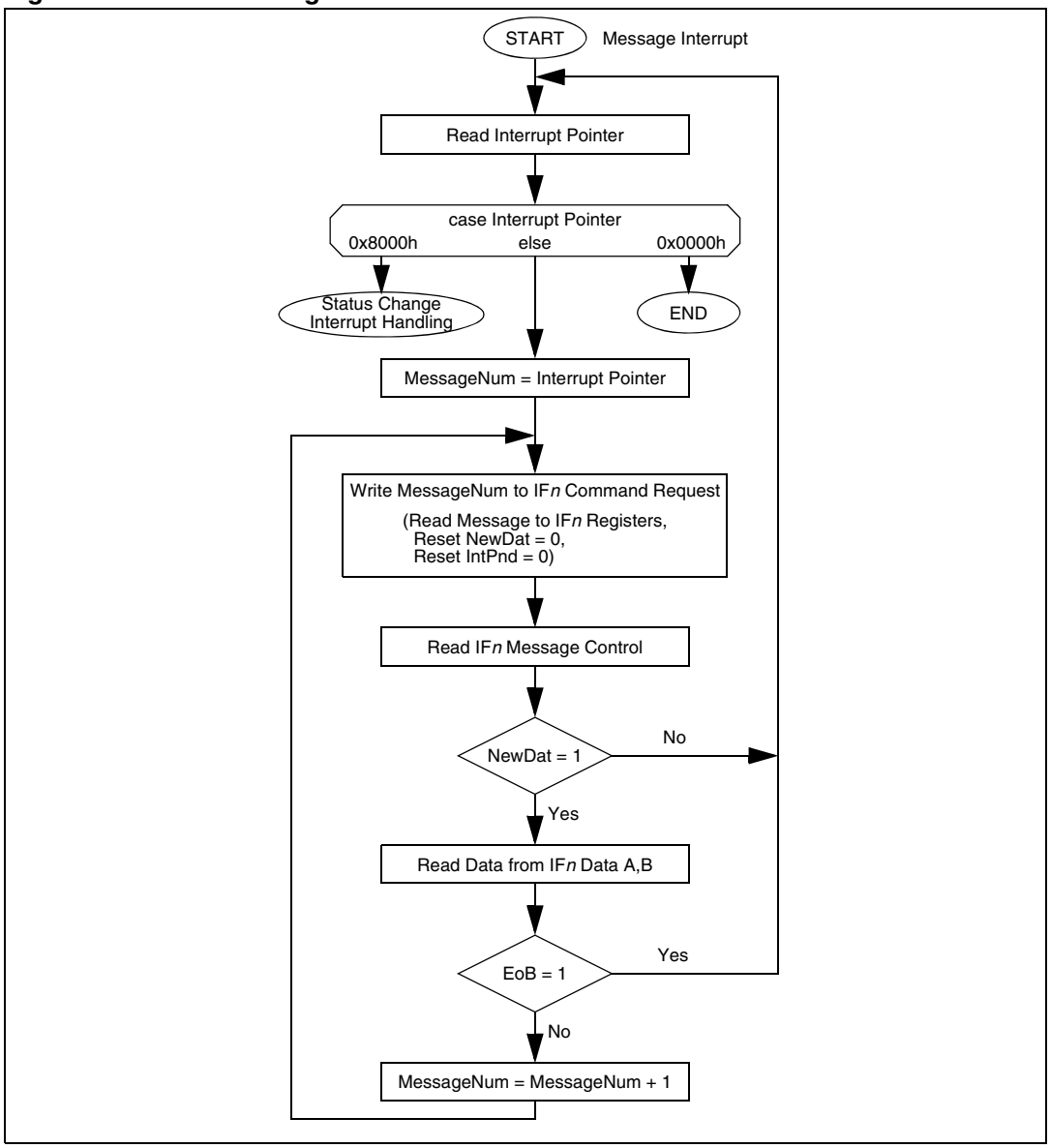
Reading from a FIFO buffer

When the CPU transfers the contents of a Message Object to the IFn Message Buffer register by writing its number to the IFn Command Request Register, the corresponding Command Mask Register should be programmed in such a 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 the Message Objects starting at the FIFO Object with the lowest message number.

Figure 53 shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.

Figure 53. CPU handling of a FIFO buffer



16.7.9 Handling 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 application software has cleared it.

The Status Interrupt has the highest priority. Among the message interrupts, interrupt priority of the Message Object decreases with increasing message number.

A message interrupt is cleared by clearing the IntPnd bit of the Message Object. 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 IRQ interrupt signal to the EIC is active. The interrupt 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 0x8000 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 1 & 2 (CAN_IPnR) on page 206*).

An interrupt service routine that is 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.

16.7.10 Configuring 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. However, in the case of arbitration, 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 interaction of the CAN nodes on the CAN bus.

Bit time and bit rate

CAN supports bit rates 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 (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the oscillator periods of the CAN nodes (f_{osc}) may be different.

The frequencies of these oscillators 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 bit rates by re-synchronizing to the bit stream.

According to the CAN specification, the bit time is divided into four segments (see *Figure 54*). 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 43*). 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_{APB} and the BRP bit of the Bit Timing Register (CAN_BTR): t_q = BRP / f_{APB}.

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 re-synchronization may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

Figure 54. Bit timing

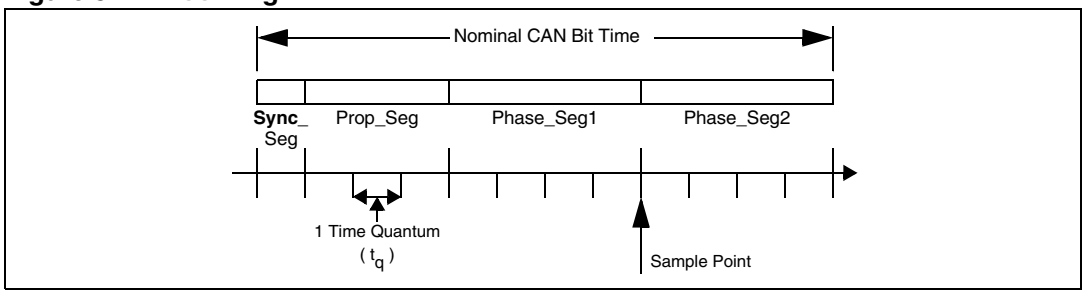


Table 43. CAN bit time parameters

Parameter	Range	Remark
BRP	[1 .. 32]	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 Segment
This table describes the minimum programmable ranges required by the CAN protocol		

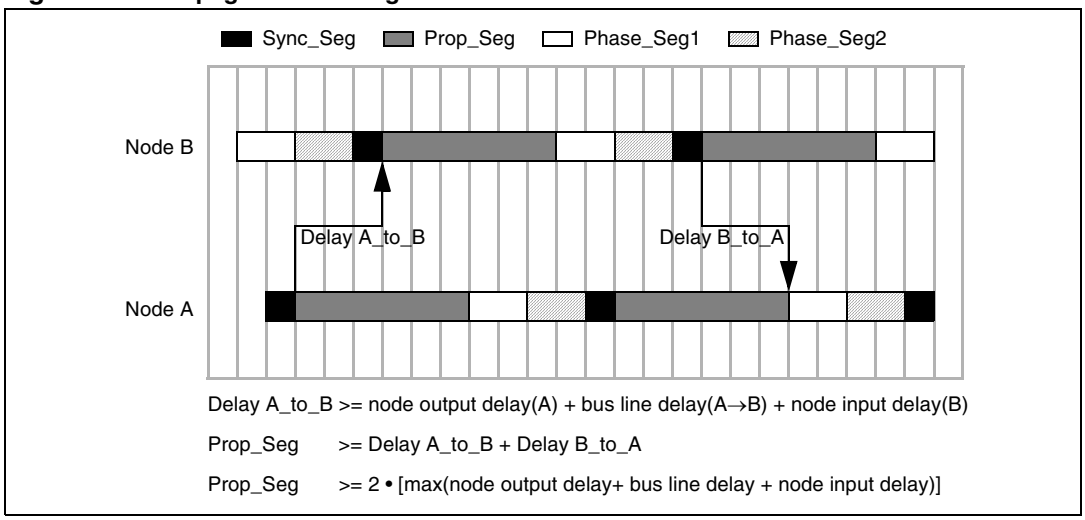
A given bit rate may be met by different bit time configurations, but for the proper function of the CAN network the physical delay times and the oscillator's tolerance range have to be considered.

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 bitwise arbitration and the dominant acknowledge bit provided by receivers of CAN messages requires 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 55* shows the phase shift and propagation times between two CAN nodes.

Figure 55. Propagation time segment



In this example, both nodes A and B are transmitters, performing an arbitration for the CAN bus. 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 respect 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 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 arrives at node A after the start of Phase_Seg1, it can 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 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 to short) that causes sporadic bus errors.

Some CAN implementations provide an optional 3 Sample Mode but 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 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 bus level at the actual time quantum 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 Re-synchronization.

A Hard Synchronization is done once at the start of a frame and inside a frame only when Re-synchronizations 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 Re-synchronization
  Re-synchronization 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 Re-synchronization 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 Re-synchronization 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 Re-synchronization are the same. If the magnitude of the phase error is larger than SJW, the Re-synchronization cannot compensate the phase error completely, an error (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).

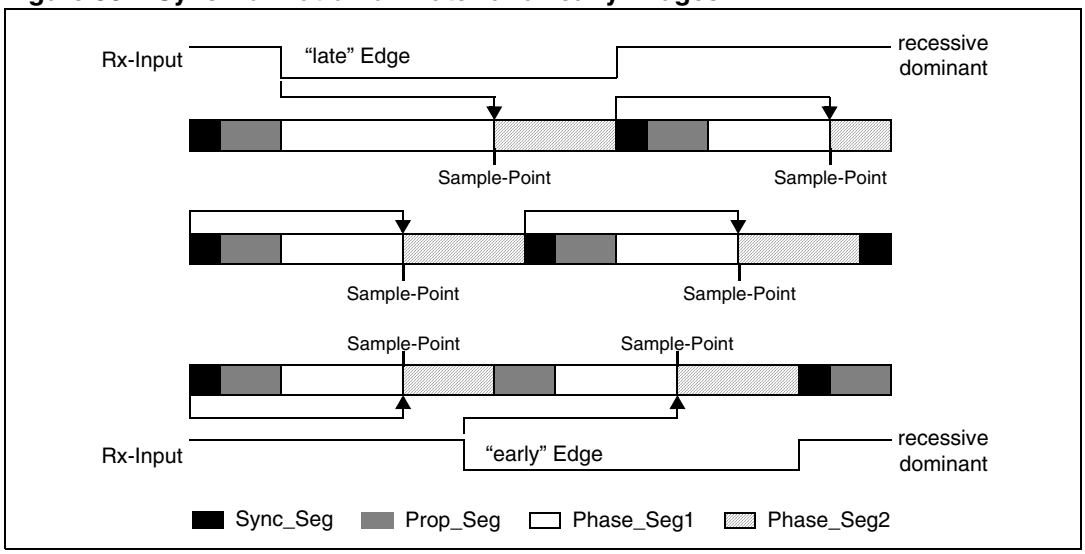
Apart from noise spikes, most synchronizations are caused by arbitration. All nodes synchronize “hard” on the edge transmitted by the “leading” transceiver that started transmitting first, but due to propagation delay times, they cannot become ideally synchronized. The “leading” transmitter does not 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 tolerance range.

The examples in *Figure 56* 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 56. 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.

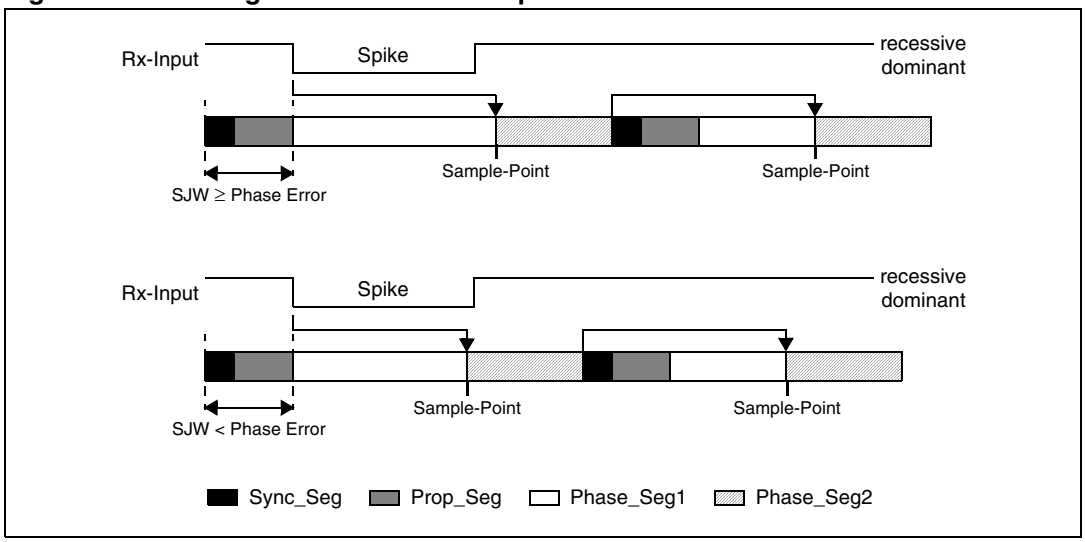
In the second example, an edge from recessive to dominant occurs during Phase_Seg2. The edge is “early” since it occurs before a Sync_Seg. Reacting to the “early” edge, Phase_Seg2 is shortened and Sync_Seg is omitted, so that the distance from the edge to the Sample Point is the same as it would have been from an Sync_Seg to the Sample Point if no edge had occurred. As in the previous example, the magnitude of phase error of this “early” edge’s 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 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 57* 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 57. 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.

Oscillator tolerance range

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 only CAN controllers to implement protocol version 1.1 have been Intel 82526 and Philips 82C200, both are superseded by successor products. The protocol update to version 2.0 (A and B) had no influence on the oscillator tolerance.

The tolerance range df for an oscillator frequency f_{osc} around the nominal frequency f_{nom} is:

$$(1 - df) \bullet f_{nom} \leq f_{osc} \leq (1 + df) \bullet f_{nom}$$

It depends on the proportions of Phase_Seg1, Phase_Seg2, SJW, and the bit time. The maximum tolerance df is the defined by two conditions (both shall be met):

$$\text{I:} \quad df \leq \frac{\min(Phase\_Seg1,\ Phase\_Seg2)}{2 \cdot (13 \cdot bit\_time - Phase\_Seg2)}$$

$$\text{II:} \quad df \leq \frac{SJW}{20 \cdot bit\_time}$$

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 that 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 oscillator tolerance of 1.58%. 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 bit rates of up to 125 kBit/s (bit time = 8 µs) with a bus length of 40 m.

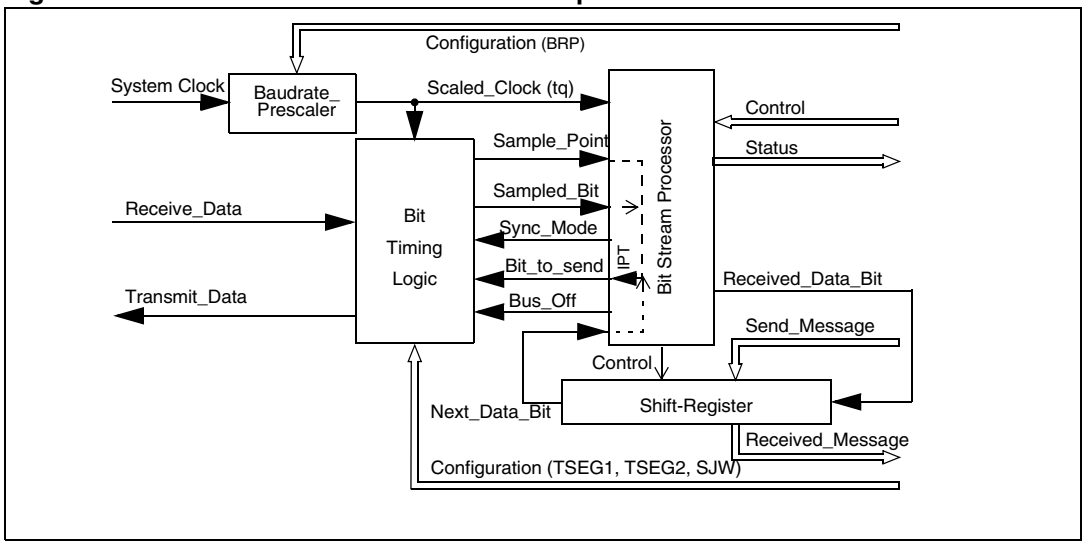
Configuring the CAN protocol controller

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 58 on page 223*).

In these bit timing registers, the four components TSEG1, TSEG2, SJW, and BRP have to be programmed to a numerical value that is one less than its functional value. Therefore, instead of values in the range of [1..n], values in the range of [0..n-1] are programmed. That way, e.g. 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 58. Structure of the CAN core's CAN protocol controller



The data in the bit timing registers is 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 BSP state machine is evaluated once each bit time, at the Sample Point.

The Shift Register sends the messages serially and receives the messages parallely. 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

that is needed to calculate the next bit to be sent after the Sample point(e.g. 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 IPT for the C_CAN 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.

Calculating bit timing parameters

Usually, the calculation of the bit timing configuration starts with a desired bit rate 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 IPT of the CAN controller, which, depending on the actual implementation, is 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 *Oscillator tolerance range on page 222*

If more than one configuration is possible, that configuration allowing the highest oscillator tolerance range should be chosen.

CAN nodes with different system clocks require different configurations to come to the same bit rate. 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 oscillator tolerance range of the CAN systems is limited by that node with the lowest tolerance range.

The calculation may show that bus length or bit rate have to be decreased or that the stability of the oscillator frequency 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)&

(SynchronisationJumpWidth-1)&(Prescaler-1)

Example for bit timing at high baudrate

In this example, the frequency of APB_CLK is 10 MHz, BRP is 0, the bit rate is 1 MBit/s.

t_q	100	ns	$= t_{APB\_CLK}$
delay of bus driver	50	ns	
delay of receiver circuit	30	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	600	ns	$= 6 \bullet t_q$
t_{SJW}	100	ns	$= 1 \bullet t_q$
t_{TSeg1}	700	ns	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	200	ns	= Information Processing Time + $1 \bullet t_q$
$t_{Sync-Seg}$	100	ns	$= 1 \bullet t_q$
bit time	1000	ns	$= t_{Sync-Seg} + t_{TSeg1} + t_{TSeg2}$
			$= \frac{\min(PB1,PB2)}{2x(13xbit\_time-PB2)}$
tolerance for APB_CLK	0.39	%	
			$= \frac{0.1\mu s}{2x(13x1\mu s-0.2\mu s)}$

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 is programmed to= 0x1600.

Example for bit timing at low baudrate

In this example, the frequency of APB_CLK is 2 MHz, BRP is 1, the bit rate is 100 KBit/s.

t_q	1	µs	$= 2 \bullet t_{APB\_CLK}$
delay of bus driver	200	ns	
delay of receiver circuit	80	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	1	µs	$= 1 \bullet t_q$
t_{SJW}	4	µs	$= 4 \bullet t_q$
t_{TSeg1}	5	µs	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	4	µs	= Information Processing Time + $3 \bullet t_q$
$t_{Sync-Seg}$	1	µs	$= 1 \bullet t_q$
bit time	10	µs	$= t_{Sync-Seg} + t_{TSeg1} + t_{TSeg2}$
			$= \frac{\min(PB1,PB2)}{2x(13xbit\_time-PB2)}$
tolerance for APB_CLK	1.58	%	
			$= \frac{4\mu s}{2x(13x10\mu s-4\mu s)}$

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 is programmed to= 0x34C1.

17 I²C interface module (I2C)

An I²C Bus Interface serves as an interface between the microcontroller and the serial I²C bus. It provides both multimaster and slave functions, and controls all I²C bus-specific sequencing, protocol, arbitration and timing. It supports fast I²C mode (400kHz).

17.1 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 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:

- Start bit detection flag
- 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

17.2 General description

In addition to receiving and transmitting data, this interface converts them 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.

17.2.1 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.

17.2.2 Communication flow

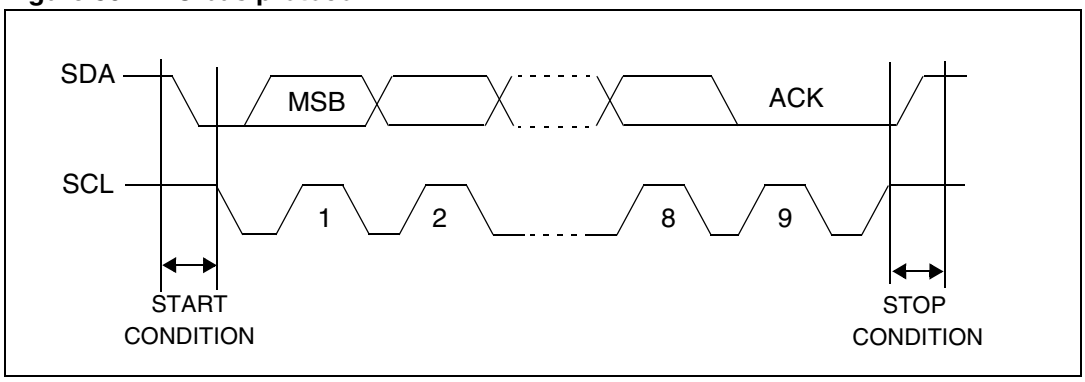
In Master mode, it initiates a data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by hardware as soon as the Master mode is selected.

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 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to *Figure 59*.

Figure 59. 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).

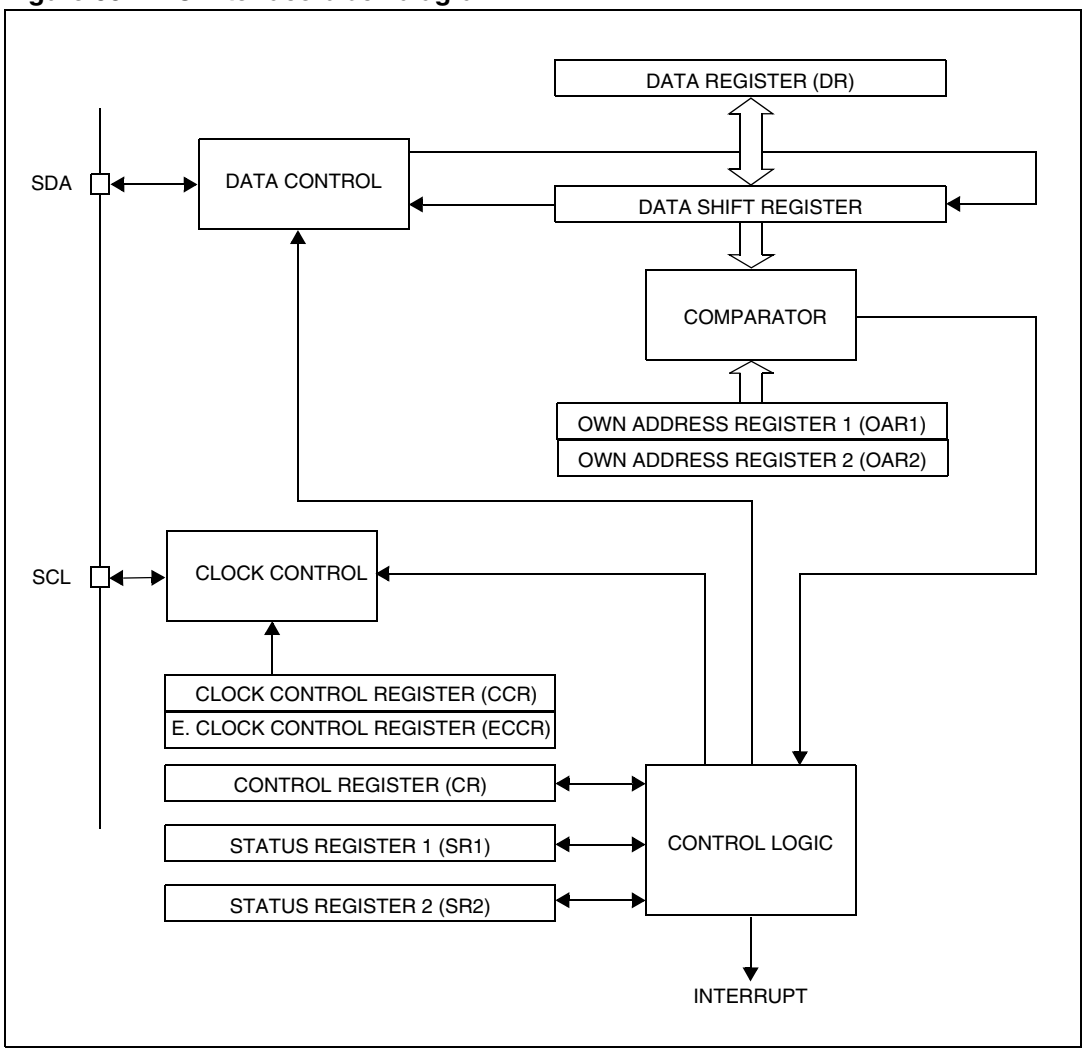
17.2.3 SDA/SCL line control

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 I^2C bus mode.

Figure 60. I^2C interface block diagram



17.3 Functional description

Refer to the I2Cn_CR, I2Cn_SR1 and I2Cn_SR2 registers in *Section 17.5* for the bit definitions.

By default the I^2C 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 FRi bits in the I2Cn_OAR2 register.

17.3.1 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 I2Cn_SR1 register, **holding the SCL line low** (see *Figure 61* Transfer sequencing EV1).
Next, in 7-bit mode read the I2Cn_DR register to determine from the least significant bit (Data Direction Bit) if the slave must enter Receiver or Transmitter mode.

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).

17.3.2 Slave receiver

Following the address reception and after I2Cn_SR1 register has been read, the **slave receives bytes from the SDA line into the** I2Cn_DR register **via t**he 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 I2Cn_SR1 register followed by a read of the I2Cn_DR register, **holding the SCL line low** (see *Figure 61* Transfer sequencing EV2).

17.3.3 Slave transmitter

Following the address reception and after I2Cn_SR1 register has been read, **the slave sends bytes from** the I2Cn_DR register to **the SDA line** via **t**he internal shift register.

The slave waits for a read of the I2Cn_SR1 register followed by a write in the I2Cn_DR register, **holding the SCL line low** (see *Figure 61* 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.

17.3.4 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 I2Cn_SR2 register (see *Figure 61* Transfer sequencing EV4).

17.3.5 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, released 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.

17.3.6 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.

17.3.7 Master mode

To switch from default Slave mode to Master mode a Start condition generation is needed.

17.3.8 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 read of the I2Cn_SR1 register followed by a write in the I2Cn_DR register with the Slave address, **holding the SCL line low** (see *Figure 61* Transfer sequencing EV5).

17.3.9 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 bit is set by hardware with interrupt generation if the ITE bit is set.

Then the master waits for a read of the I2Cn_SR1 register followed by a write in the I2Cn_DR register, **holding the SCL line low** (see *Figure 61* 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 bit is set by hardware with interrupt generation if the ITE bit is set.

Then the master waits for a read of the I2Cn_SR1 register followed by a write in the I2Cn_CR register (for example set PE bit), **holding the SCL line low** (see *Figure 61* 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).

17.3.10 Master receiver

Following the address transmission and after I2Cn_SR1 and I2Cn_CR registers have been accessed, the **master receives bytes from the SDA line into the** I2Cn_DR register **via t**he 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 SR1 register followed by a read of the I2Cn_DR register, **holding the SCL line low** (see *Figure 61* Transfer sequencing EV7).

To close the communication: before reading the last byte from the I2Cn_DR 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.

17.3.11 Master transmitter

Following the address transmission and after I2Cn_SR1 register has been read, **the master sends bytes from** the I2Cn_DR register to **the SDA line** via **t**he internal shift register.

The master waits for a read of the I2Cn_SR1 register followed by a write in the I2Cn_DR register, **holding the SCL line low** (see *Figure 61* 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.

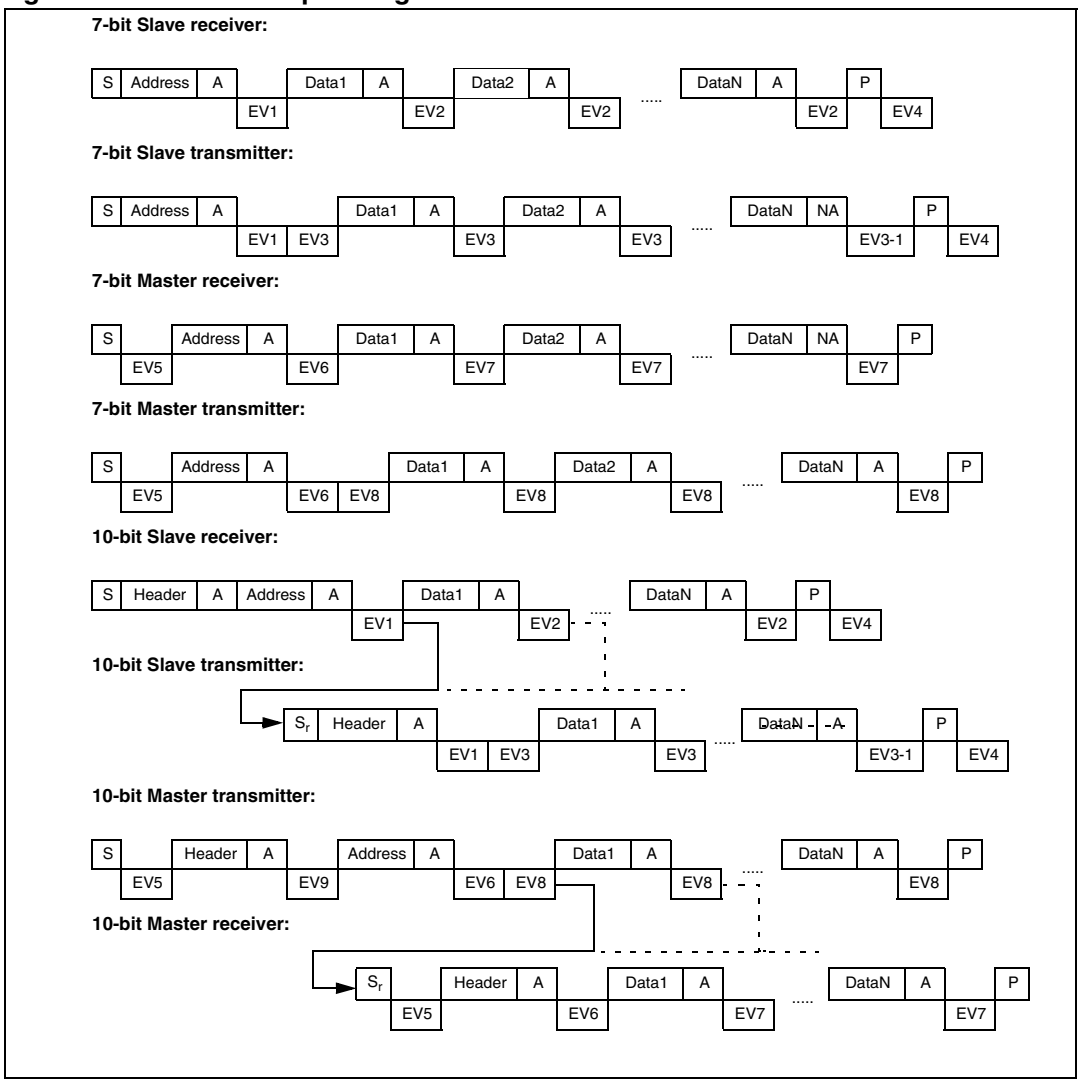
To close the communication: after writing the last byte to the I2Cn_DR register, set the STOP bit to generate the Stop condition. The interface goes automatically back to slave mode (M/SL bit cleared).

17.3.12 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.
  In this case 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.

Figure 61. 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 I2Cn_SR1 register.

EV2: EVF=1, BTF=1, cleared by reading I2Cn_DR register.

EV3: EVF=1, BTF=1, cleared by reading I2Cn_SR1 register followed by writing to the DR register.

EV3-1: EVF=1, AF=1, BTF=1; AF is cleared by reading SR2 register. BTF is cleared by releasing the lines (STOP=1, STOP=0) or by writing I2Cn_DR register (DR=FFh).

Note: If lines are released bySTOP=1, STOP=0, the subsequent EV4 is not seen.

EV4: EVF=1, STOPF=1, cleared by reading SR2 register.

EV5: EVF=1, SB=1, cleared by reading I2Cn_SR1 register followed by writing I2Cn_DR register.

EV6: EVF=1, ENDAD=1 cleared by reading I2Cn_SR2 register followed by writing I2Cn_CR register (for example PE=1).

EV7: EVF=1, BTF=1, cleared by reading the I2Cn_DR register.

EV8: EVF=1, BTF=1, cleared by writing to the I2Cn_DR register.

EV9: EVF=1, ADD10=1, cleared by reading the I2Cn_SR1 register followed by writing to the I2Cn_DR register.

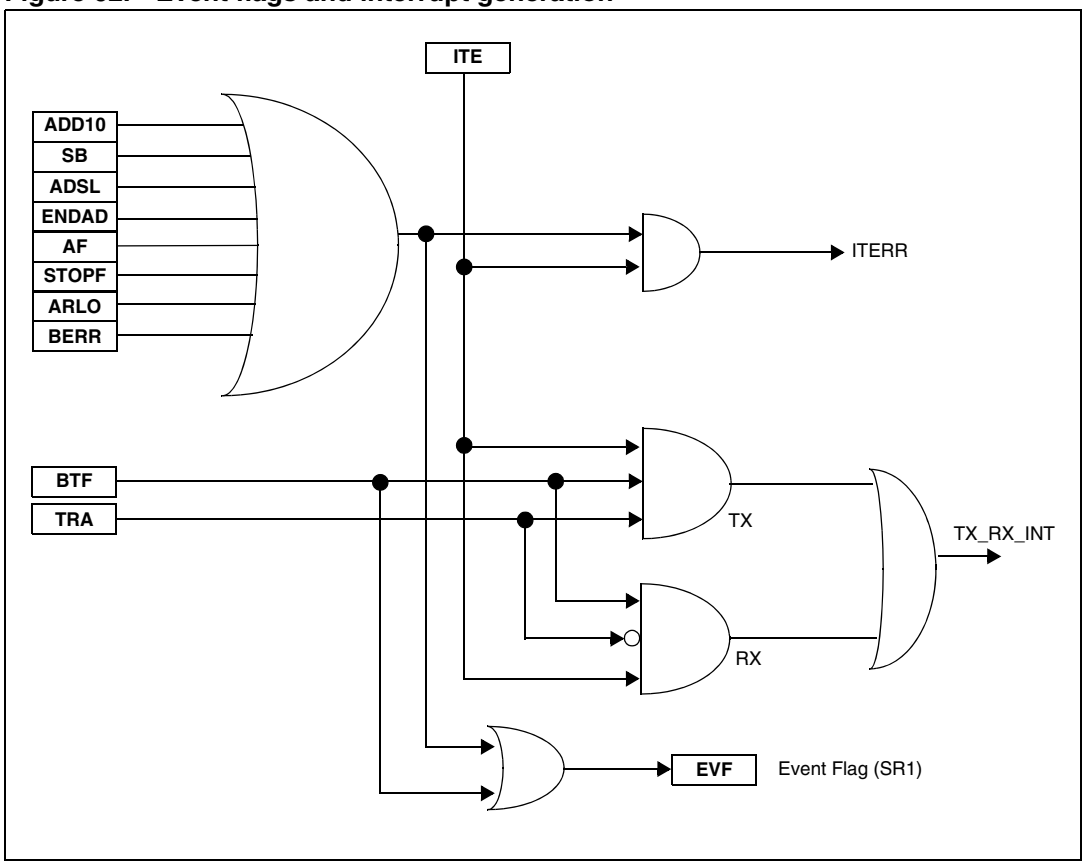
17.4 Interrupts

Several interrupt events can be flagged by the module:

- requests related to bus events, like start or stop events, arbitration lost, etc.;
- requests related to data transmission and/or reception;

These requests are issued to the interrupt controller by two different lines as described in *Figure 62*. The different flags identify the events and can be polled by the software (interrupt service routine).

Figure 62. Event flags and interrupt generation



17.5 Register description

17.5.1 I²C Control Register (I2Cn_CR)

Address Offset: 00h
Reset value: 00h

7	6	5	4	3	2	1	0
reserved		PE	ENGC	START	ACK	STOP	ITE
-		rw	rw	rw	rw	rw	rw

Bits 7:6 = Reserved, always return ‘0’ when read.

Bit 5 = **PE:** *Peripheral Enable.*
This bit is set and cleared by software.
0: Peripheral disabled
1: Master/Slave capability

Note: 0: all the bits of the I2Cn_CR register and the I2Cn_SR register except the STOP bit are reset. All outputs are released while PE=0.

1: the corresponding I/O pins are selected by hardware as alternate functions.

To enable the I²C interface, write the I2Cn_CR register **TWICE** with PE=1 as the first write only activates the interface (only PE is set).

Bit 4 = **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.

Bit 3 = **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 2 = **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

Bit 1 = **STOP**: *Generation of a Stop condition.*
This bit is set and cleared by software. It is also cleared by hardware in master mode. Note: 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.

Bit 0 = **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 62* for the relationship between the events and the interrupts.
SCL is held low when the ADD10, SB, BTF or ADSL flags or an EV6 event (See *Figure 61*) is detected.

17.5.2 I2C Status Register 1 (I2Cn_SR1)

Address Offset: 04h
Reset value: 00h

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 7 = **EVF**: *Event flag.*
This bit is set by hardware as soon as an event occurs. It is cleared by software reading I2Cn_SR2 register in case of error event or as described in *Figure 61*. 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)
- 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).

Bit 6 = **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 I2Cn_SR2 register followed by a write in the I2Cn_DR 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).

Bit 5 = **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.

Bit 4 = **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

Bit 3 = **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 I2Cn_SR1 register followed by a read or write of I2Cn_DR 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 61*). BTF is cleared by writing the next byte in I2Cn_DR register.
- Following a byte reception, this bit is set after transmission of the acknowledge clock pulse if ACK=1. BTF is cleared by reading I2Cn_SR1 register followed by reading the byte from I2Cn_DR register.

The SCL line is held low while BTF=1.
0: Byte transfer not done
1: Byte transfer succeeded

Bit 2 = **ADSL**: *Address matched (Slave mode).* This bit is set by hardware as soon as the received slave address matched with the I2Cn_OAR register content or a general call is recognized. An interrupt is generated if ITE=1. It is cleared by software reading I2Cn_SR1 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 1 = **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.

Bit 0 = **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 I2Cn_SR1 register followed by writing the address byte in I2Cn_DR register. It is also cleared by hardware when the interface is disabled (PE=0).
0: No Start condition.
1: Start condition generated.

17.5.3 I2C Status Register 2 (I2Cn_SR2)

Address Offset: 08h
Reset value: 00h

7	6	5	4	3	2	1	0
reserved		ENDAD	AF	STOPF	ARLO	BERR	GCAL
-		r	r	r	r	r	r

Bits 7:6 = Reserved, always return ‘0’ when read.

Bit 5 = **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 address once again; hence the ENDAD flag is set, without waiting for the LSB of the address.It is cleared by software by reading SR2 and a following write to the CR or by hardware when the interface is disabled (PE=0).
0: No end of address transmission
1: End of address transmission

Bit 4 = **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 I2Cn_SR2 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

Bit 3 = **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 I2Cn_SR2 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 2 = **ARLO**: *Arbitration lost*.
This bit is set by hardware when the interface loses the arbitration of the bus to another master. An interrupt is generated if ITE=1. It is cleared by software reading I2Cn_SR2 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

Bit 1 = **BERR**: *Bus error.*
This bit is set by hardware when the interface detects a misplaced Start or Stop condition. An interrupt is generated if ITE=1. It is cleared by software reading I2Cn_SR2 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

Bit 0 = **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

17.5.4 I2C Clock Control Register (I2Cn_CCR)

Address Offset: 0Ch
Reset value: 00h

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 7 = **FM/SM**: *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

Bit 6:0 = **CC[6:0]**: *12-bit clock divider*.
These bits along with CC[11:7] of the Extended Clock Control Register 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): $f_{SCL} \leq 100kHz$
  $f_{SCL} = f_{MCLK}/ (2 \times (CC[11:7]+7))$
  Given a certain f_{MCLK} is easy to obtain the right divider factor:
  $CC[11:0] = (f_{MCLK} / (2*f_{SCL})) - 7 = (t_{SCL} / (2*t_{MCLK})) - 7$
- Fast mode (FM/SM=1): $100kHz < f_{SCL} < 400kHz$
  $f_{SCL} = f_{MCLK}/ (3 \times (CC[11:0]+9))$
  Given a certain f_{MCLK} is easy to obtain the right divider factor:
  $CC[11:0] = (f_{MCLK}/ (3*f_{SCL})) - 9= (t_{SCL} / 3*t_{MCLK})) - 9$

Note: The programmed f_{SCL} assumes no load on SCL and SDA lines.

Note: For a correct usage of the divider, CC[11:0] must be equal or greater than 0x002 (000000000010b). CC[11:0] equal to 0x001 (000000000001b) is not admitted.

17.5.5 I2C Extended Clock Control Register (I2Cn_ECCR)

Address Offset: 1Ch
Reset value: 00h

7	6	5	4	3	2	1	0
reserved			CC11	CC10	CC9	CC8	CC7
			rw	rw	rw	rw	rw

Bits 7:5 = Reserved, always return ‘0’ when read.

Bits 6:0 = **CC[11:7]**: *12-bit clock divider.*
These bits along with those of the Clock Control Register select the speed of the bus (f_{SCL}) depending on the I^2C mode. They are not cleared when the interface is disabled (PE=0)

17.5.6 I2C Own Address Register 1 (I2Cn_OAR1)

Address Offset: 10h
Reset value: 00h

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

Bits 7:1 = **ADD[7:1]**: *Interface address*.
These bits define the I^2C bus address of the interface. They are not cleared when the interface is disabled (PE=0).

Bit 0 = **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.

10-bit Addressing Mode

Bits 7:0 = **ADD[7:0]**: *Interface address*.
These are the least significant bits of the I^2C bus address of the interface. They are not cleared when the interface is disabled (PE=0).

17.5.7 I2C Own Address Register 2 (I2Cn_OAR2)

Address Offset: 14h
Reset value: 20h

7	6	5	4	3	2	1	0
FR2	FR1	FR0	reserved		ADD9	ADD8	res.
rw	rw	rw	-		rw	rw	-

Bits 7:5 = **FR[2:0]**: *Frequency bits.*
These bits are set by software only when the interface is disabled (PE=0). To configure the interface to I^2C specified delays select the value corresponding to the system frequency f_{MCLK}.

Table 44. Frequency bits

f_{MCLK} Range (MHz)	FR2	FR1	FR0
5 - 10	0	0	0
10 - 16.67	0	0	1
16.67 - 26.67	0	1	0
26.67 - 40	0	1	1
40 - 53.33	1	0	0
53.33 -66	1	0	1
66 - 80	1	1	0
80 - 100	1	1	1

Bits 4:3 = Reserved, always return ‘0’ when read.

Bits 2:1 = **ADD[9:8]**: *Interface address*.
These are the most significant bits of the I^2C bus address of the interface (10-bit mode only). They are not cleared when the interface is disabled (PE=0).

Bit 0 = Reserved, always returns ‘0’ when read.

17.5.8 I2C Data Register (I2Cn_DR)

Address Offset: 18h
Reset value: 00h

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 = **D[7:0]**: *8-bit Data Register.*
These bits contain the byte to be received or transmitted on the bus.

- Transmitter mode: Byte transmission start automatically when the software writes in the I2Cn_DR register.
- Receiver mode: the first data byte is received automatically in the I2Cn_DR register using the least significant bit of the address. Then, the following data bytes are received one by one after reading the I2Cn_DR register.

17.6 I2C register map

Table 45. I^2C interface register map

Address Offset	Register Name	7	6	5	4	3	2	1	0
00h	I2Cn_CR	reserved		PE	ENGC	START	ACK	STOP	ITE
04h	I2Cn_SR1	EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB
08h	I2Cn_SR2	reserved		ENDAD	AF	STOPF	ARLO	BERR	GCAL
0Ch	I2Cn_CCR	FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0
10h	I2Cn_OAR1	ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0
14h	I2Cn_OAR2	FR2	FR1	FR0	reserved		ADD9	ADD8	res.
18h	I2Cn_DR	D7	D6	D5	D4	D3	D2	D1	D0
1Ch	I2Cn_ECCR	reserved			CC11	CC10	CC9	CC8	CC7

See *Table 2 on page 17* for base addresses.

18 Buffered SPI (BSPI)

The BSPI block is a standard 4-pin Serial Peripheral Interface for inter-IC control communication. It interfaces on one side to the SPI bus and on the other has a standard register data and interrupt interface.

The BSPI contains two 16-word x 16-bit FIFO's one for receive and the other for transmit. The BSPI can directly operate with words 8 and 16 bit long and can generates interrupts or DMA requests separately for receive and transmit events.

18.1 Main features

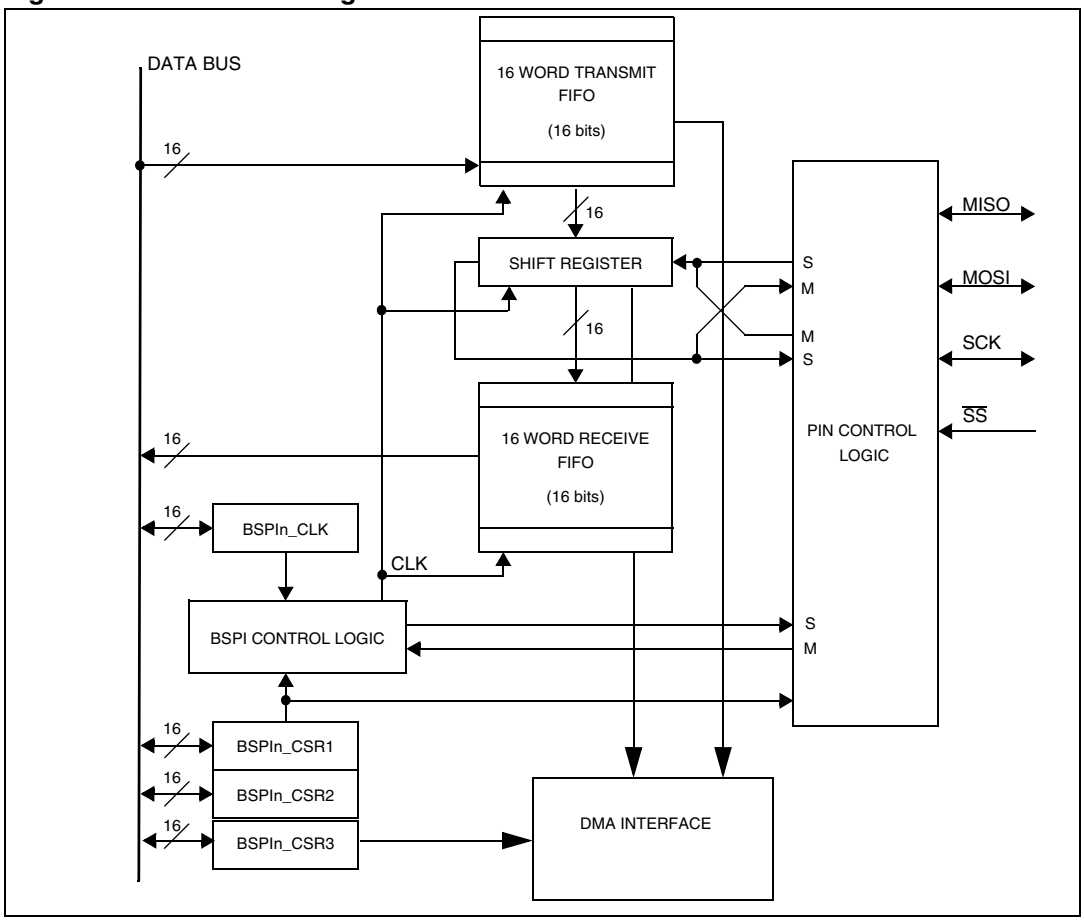
- Programmable depth receive FIFO
- Maximum 16 word receive FIFO
- Programmable depth transmit FIFO
- Maximum 16 word transmit FIFO
- Master and Slave modes supported
- Internal clock prescaler
- Programmable DMA interface
- Bandwidth of up to 6 Mb/s at 36 MHz

18.2 Functional description

The processor views the BSPI as a memory mapped peripheral, which may be used by standard polling, interrupt programming techniques or DMA controlled access. Memory-mapping means processor communication can be achieved using standard instructions and addressing modes.

When an SPI transfer occurs data is transmitted and received simultaneously. A serial clock line synchronizes shifting and sampling of the information on the two serial data lines. A slave select line allows individual selection of a slave device. The central elements in the BSPI system are the 16-bit shift register and the read data buffer which is 16 words x 16-bit.A BSPI-DMA interface is also present to allow for data to be transferred to/from memory using the DMA. A block diagram of the BSPI is shown in *Figure 63*.

Figure 63. BSPI block diagram



18.2.1 BSPI pin description

The BSPI is a four wire, bi-directional bus. The data path is determined by the mode of operation selected. A master and a slave mode are provided together with the associated pad control signals to control pad direction. These pins are described in *Table 46*.

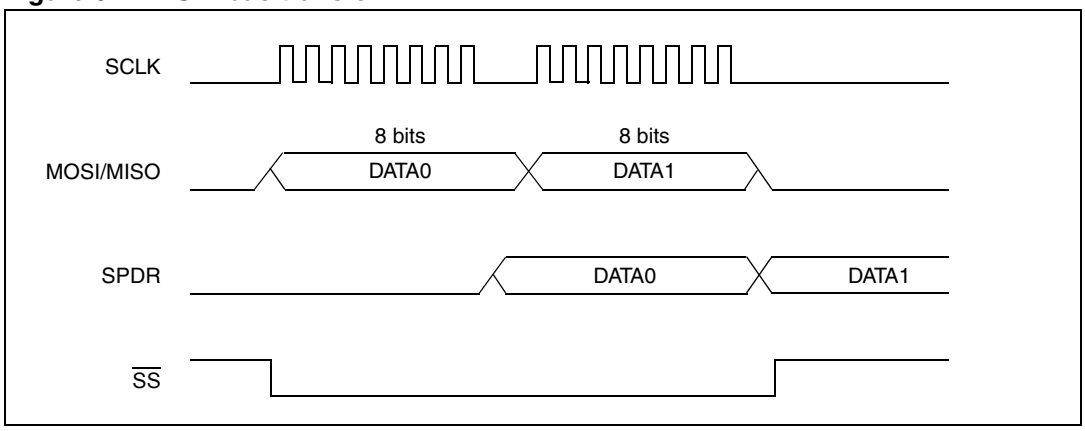
Table 46. BSPI pins

Pin Name	Description
SCLK	The bit clock for all data transfers. When the BSPI is a master the SCLK is output from the chip. When configured as a slave the SCLK is input from the external source.
MISO	Master Input/Slave Output serial data line.
MOSI	Master Output/Slave Input serial data line.
SS	Slave Select. The $\overline{SS}$ input pin is used to select a slave device. Must be pulled low after the SCLK is stable and held low for the duration of the data transfer. The $\overline{SS}$ on the master must be deasserted high.This signal can be masked when in master mode-see register description of CSR Reg3 bit 0

18.2.2 BSPI operation

During a BSPI transfer (see *Figure 64*), data is shifted out and shifted in (transmitted and received) simultaneously. The SCLK line synchronizes the shifting and sampling of the information. It is an output when the BSPI is configured as a master and an input when the BSPI is configured as a slave. Selection of an individual slave BSPI device is performed on the slave select line and slave devices that are not selected do not interfere with the BSPI buses.

Figure 64. BSPI bus transfer



The CPOL (clock polarity) and CPHA (clock phase) bits of the BSPIn_CSR1 are used to select any of the four combinations of serial clock (see *Figure 65*, *Figure 66*, *Figure 67* and *Figure 68*). These bits must be the same for both the master and slave BSPI devices. The clock polarity bit selects either an active high or active low clock but does not affect transfer format. The clock phase bit selects the transfer format.

There is a 16-bit shift register which interfaces directly to the BSPI bus lines. As transmit data goes out from the register, received data fills the register.

Figure 65. BSPI clocking scheme (CPOL=0, CPHA=0)

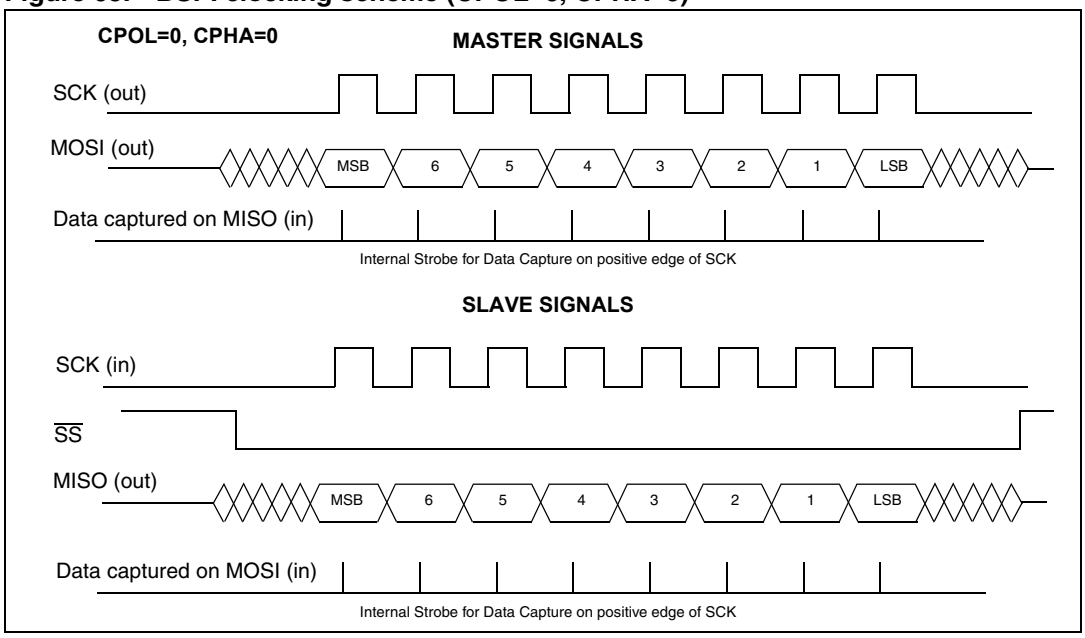


Figure 66. BSPI clocking scheme (CPOL=0, CPHA=1)

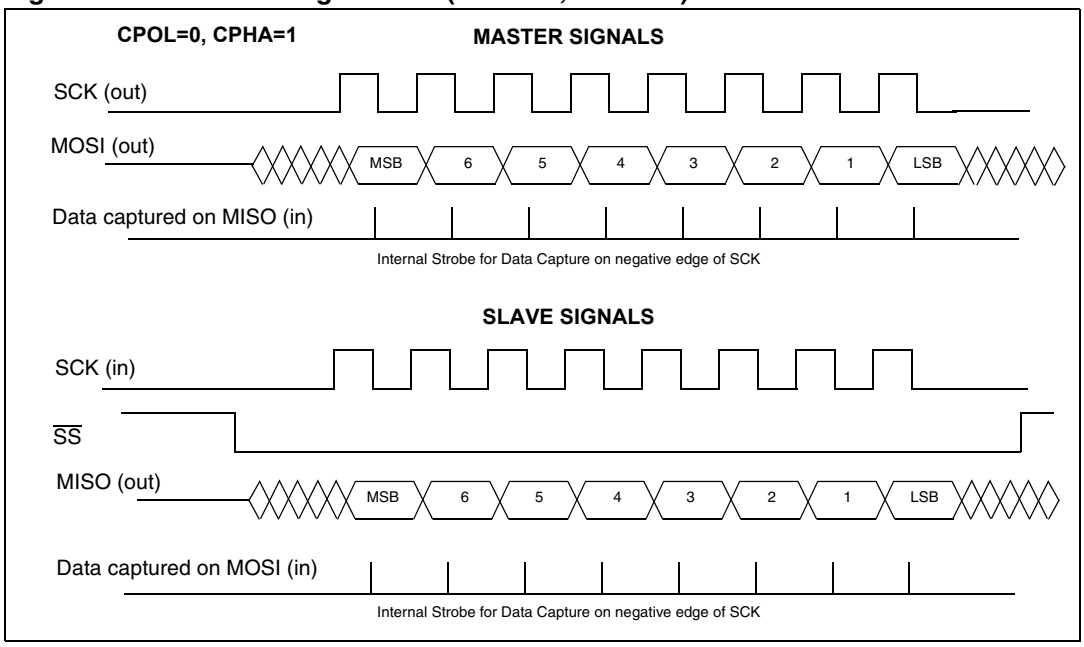


Figure 67. BSPI clocking scheme (CPOL=1, CPHA=0)

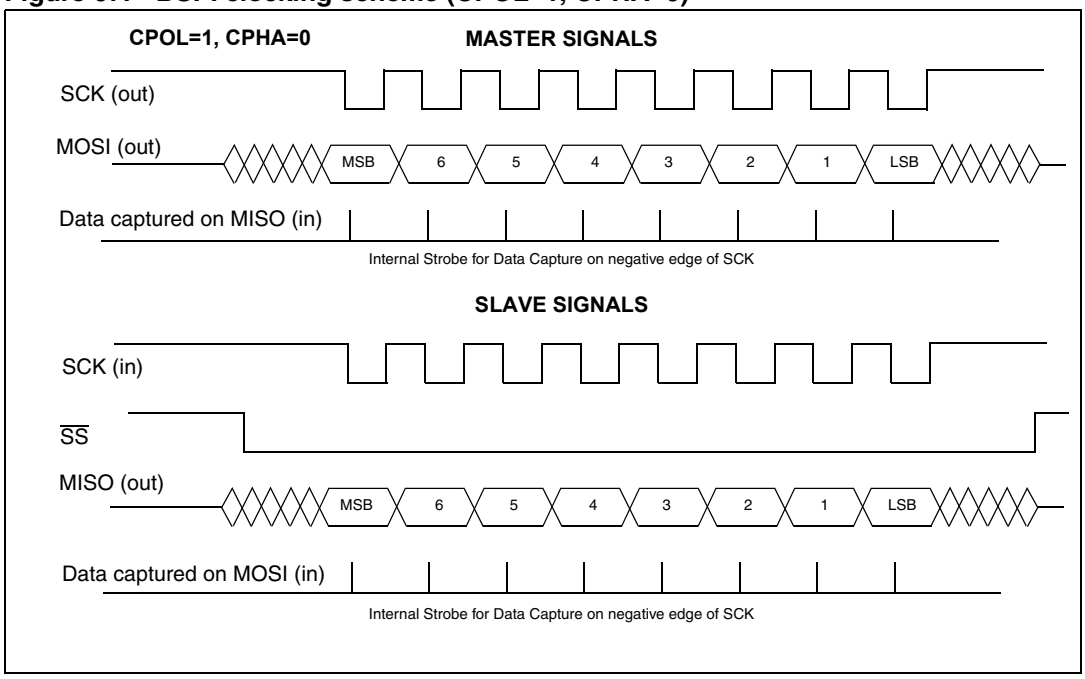
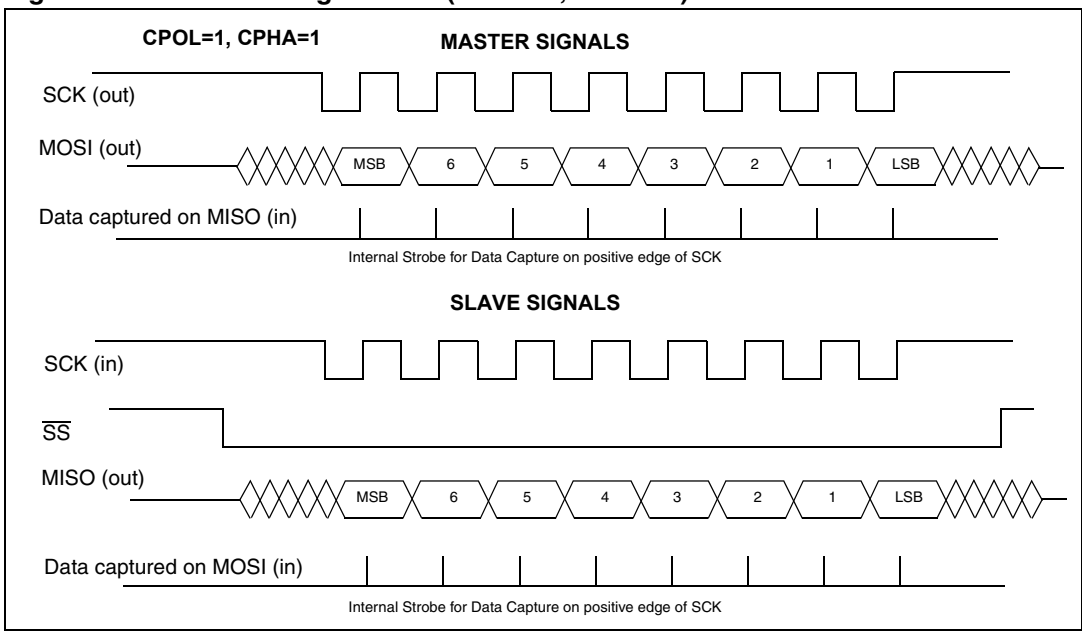


Figure 68. BSPI clocking scheme (CPOL=1, CPHA=1)



18.2.3 Transmit FIFO

The transmit FIFO consists of a 16 by 16 bit register bank which can operate in 8/16 bit modes as configured by the word length (WL[1:0]) control bits of BSPIn_CSR1. Data is left justified indicating that only the most significant portion of the word is transmitted if using 8 bit mode. After a transmission is completed the next data word is loaded from the transmit FIFO.

The user can set the depth of the FIFO from the default one location up to a maximum of sixteen locations. This can be set dynamically but will only take effect after the completion of the current transmission. Status flags report if the FIFO is full (TFF), the FIFO is not empty (TFNE), the FIFO is empty (TFE) and the transmit buffer has under flown (TUFL). The transmit interrupt enable (TIE[1:0]) control bits of BSPIn_CSR2 determine the source of the transmit interrupt. If the interrupt source is enabled then an active high interrupt will be asserted to the processor.

If the TUFL flag is asserted then a subsequent write to the transmit FIFO will clear the flag. If interrupts are enabled the interrupt will be de-asserted. The TFF and TFNE flags are updated at the end of the processor write cycle and at the end of each transmission.

18.2.4 Receive FIFO

The BSPI Receive FIFO is a 16 word by 16-bit FIFO used to buffer the data words received from the BSPI bus.

The FIFO can operate in 8-bit and 16-bit modes as configured by the WL[1:0] bits of the BSPIn_CSR1 register. Irrelevant of the word depth in the FIFO, if operating in 8-bit mode, the data will occupy both the Most Significant and the Least Significant Bytes of each location of the FIFO (data can be used either as left or right justified).

The receive FIFO enable bits RFE[3:0] declare how many words deep the FIFO is for all transfers. The FIFO defaults to one word deep. Whenever there is at least one block of data in the FIFO the RFNE bit is set in the BSPIn_CSR2 register, i.e there is data in at least one

location. The RFF flag does not get set until all locations of the FIFO contain data, i.e. RFF is set when the depth of FIFO is filled and nothing has been read out.

If the FIFO is one word deep then the RFNE and RFF flags are set once data is written to it. When the data is read then both flags are cleared. A write to and a read from the FIFO can happen independent of each once RFF is not set, if RFF is set a read must occur before the next write or an overflow (ROFL) will occur.

18.2.5 Start-up status

If the BSPI is to operate in Master mode, the MSTR bit must be set high and then the BSPI must be enabled. The TFE flag will be set, signalling that the Transmit FIFO is empty, if the TIE is set, a TFE interrupt will be generated. The data to be transferred must be written to the Transmit Data Register, the TFE interrupt will be cleared and then the BSPI clock will be generated according to the value of the BSPIn_CLK register. The Transfer of data then begins. A second TFE interrupt occurs so that the peripheral has a full data transfer time to request the data before the next transfer is to begin.

If the BSPI is to operate in slave mode, once again the device must be enabled. The $\overline{SS}$ line must only be asserted low after the SCK from the master is stable. The TFE flag will be set, depending on the BSPI being enabled, signalling that the Transmit Data register is empty and will be cleared by a write to the Transmit Data register. The second TFE interrupt occurs to request data for the following transfer.

18.2.6 Clocking problems and clearing of the shift-register

Should a problem arise on the clock which results in a misalignment of data in the shift register of the BSPI, it may be cleared by disabling the BSPI enable. This has the effect of setting the TFE which requests data to be written to the Transmit Register for the next transfer. Clearing the BSPI enable will also reset the counter of bits received. The next block of data received will be written to the next location in the FIFO continuing on from the last good transfer, if the FIFO was just one word deep it will be written to the only location available.

18.2.7 Interrupt control

The BSPI generates one interrupt based upon the status bits monitoring the transmit and receive logic. The interrupt is acknowledged or cleared by subsequent read or write operations which remove the error or status update condition. It is the responsibility of the programmer to ascertain the source of the interrupt and then remove the error condition or alter the state of the BSPI. In the case of multiple errors the interrupt will remain active until all interrupt sources have been cleared.

For example, in the case of TFE, whenever the last word has been transferred to the transmit buffer, the TFE flag is asserted. If interrupts are enabled then an interrupt will be asserted to the processor. To clear the interrupt the user must write at least one data word into the FIFO, or disable the interrupts if this condition is valid.

18.2.8 DMA interface

The DMA interface is a feature of the BSPI that allows data to be transferred to or from system memory using a DMA controller instead of main CPU. Data can be transferred in single data accesses or in burst mode, the amount of words being selectable by the

programmer, thus making efficient more use of system bus. The BSPI DMA interface has the following features:

1. The DMA interface can be totally disabled using a bit in BSPIn_CSR3 register.
2. User programmable burst size: 1, 4, 8 or 16 words can be transferred at a time.
3. Two request lines to the DMA are generated: one dedicated to the BSPI operating in transmit mode and another dedicated to the BSPI operating in receive mode. These signals are generated in independently from each other inside the BSPI.
4. Received 8-bit data, being sent from the BSPI to the memory, is arranged in a format which is compatible both with little and big endian convention, replicating the received data on both high and low byte in BSPIn_RXR register.

The DMA interface makes use of pointers inside both the transmit and receive fifo (these being independent of one another) in order for it to decide when a DMA request to transmit or receive data can be made.

Note: There is a restriction on the burst capability of DMA interface during reception. The total number of words to be transferred to system memory via DMA must be an integer multiple of the programmed Burst Length size, since DMA interface is not capable of handling incomplete received burst transfers. For example if Burst Length size is set to 4 and at the end of received data transfer BSPI FIFO has only 3 spaces, no request would be issued. On the contrary, with an even divide, the last chunk of received data will always be equal to the programmed size of the burst length.

18.3 Register description

18.3.1 BSPI control/status register 1 (BSPIn_CSR1)

Address Offset: 08h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RFE[3:0]				WL[1:0]		CPHA	CPOL	BEIE	Reserved		REIE	RIE[1:0]		MSTR	BSPE
rw				rw		rw	rw	rw	-		rw	rw		rw	rw

Bits 15:12 = **RFE[3:0]**: *Receive FIFO Enable.*
The receive FIFO can be programmed to operate with a word depth up to 16. The receive FIFO enable bits declare how many words deep the FIFO is for all transfers. The FIFO defaults to one word deep, i.e. similar to a single data register. *Table 47* shows how the FIFO is controlled.

Table 47. Rx FIFO depth

RFE[3:0]	Depth of FIFO
0000	1st word enabled
0001	1st & 2nd words enabled
0010	1-3 words enabled
0011	1-4 words enabled

Table 47. Rx FIFO depth (continued)

RFE[3:0]	Depth of FIFO
0100	1-5 words enabled
0101	1-6 words enabled
0110	1-7 words enabled
0111	1-8 words enabled
1000	1-9 words enabled
1001	1-10 words enabled
1010	1-11 words enabled
1011	1-12 words enabled
1100	1-13 words enabled
1101	1-14 words enabled
1110	1-15 words enabled
1111	1-16 words enabled

Bits 11:10 = **WL[1:0]**: *Word Length.*
These two bits configure the word length operation of the Receive FIFO and transmit data registers.
00: 8-bit
01: 16-bit
10: Reserved
11: Reserved

Bit 9 = **CPHA**: *Clock Phase Select.*
Used with the CPOL bit to define the master-slave clock relationship. When CPHA=0, as soon as the $\overline{SS}$ goes low the first data sample is captured on the first edge of SCK. When CPHA=1, the data is captured on the second edge.

Bit 8 = **CPOL**: *Clock Polarity Select.*
When this bit is cleared and data is not being transferred, a stable low value is present on the SCK pin. If the bit is set the SCK pin will idle high. This bit is used with the CPHA bit to define the master-slave clock relationship.
0 = Active high clocks selected; SCLK idles low.
1 = Active low clocks selected; SCLK idles high.

Bit 7 = **BEIE**: *Bus Error Interrupt Enable.*
When this bit is set to a '1', an interrupt will be asserted to the processor whenever a Bus Error condition occurs.

Bits 6:5 = Reserved, must be kept at reset value (0).

Bit 4 = **REIE**: *Receive Error Interrupt Enable.*
When this bit is set to a '1' and the Receiver Overflow error condition occurs, a Receive Error Interrupt will be asserted to the processor.

Bits 3:2 = **RIE[1:0]**: *BSPI Receive Interrupt Enables.*
The RIE1:0 bits are interrupt enables which configure when the processor will be interrupted on received data. The following configurations are possible:
00: Disabled
01: Receive FIFO Not Empty

10: Reserved
11: Receive FIFO Full

Bit 1 = **MSTR**: *Master/Slave Select.*
1: BSPI is configured as a master
0: BSPI is configured as a slave

Bit 0 = **BSPE**: *BSPI System Enable.*
1: BSPI system is enabled
0: BSPI system is disabled

18.3.2 BSPI control/status register 2 (BSPIn_CSR2)

Address Offset: 0Ch

Reset value: 0040h

15 14	13 12 11 10	9	8	7	6	5	4	3	2	1	0
TIE[1:0]	TFE[3:0]	TFNE	TFF	TUFL	TFE	ROFL	RFF	RFNE	BERR	Res.	DFIFO
rw	rw	r	r	r	r	r	r	r	r	-	w

Bits 15:14 = **TIE[1:0]**: *BSPI Transmit Interrupt Enable.*
These bits control the source of the transmit interrupt:
00: Disabled
01 Transmit FIFO Empty
10: Transmit underflow
11: Transmit FIFO Full

Bits 13:10 = **TFE[3:0]**: *Transmit FIFO Enable.*
These bits control the depth of the transmit FIFO. The table below indicates all valid settings.

Table 48. Tx FIFO depth

TFE[3:0]	Depth of FIFO
0000	1st word enabled
0001	1st & 2nd words enabled
0010	1-3 words enabled
0011	1-4 words enabled
0100	1-5 words enabled
0101	1-6 words enabled
0110	1-7 words enabled
0111	1-8 words enabled
1000	1-9 words enabled
1001	1-10 words enabled
1010	1-11 words enabled

Table 48. Tx FIFO depth (continued)

TFE[3:0]	Depth of FIFO
1011	1-12 words enabled
1100	1-13 words enabled
1101	1-14 words enabled
1110	1-15 words enabled
1111	1-16 words enabled

Bit 9 = **TFNE**: *Transmit FIFO Not Empty.*
This bit is set whenever the FIFO contains at least one data word.

Bit 8 = **TFF**: *Transmit FIFO Full.*
TFF is set whenever the number of words written to the transmit FIFO is equal to the number of FIFO locations enabled by TFE[3:0]. The flag is set immediately after the data write is complete.

Bit 7 = **TUFL**: *Transmit Underflow.*
This status bit gets set if the TFE bit is set and, by the time the Transmit Data Register contents are to be transferred to the shift register for the next transmission, the processor has not yet put the data for transmission into the Transmit Data Register.
TUFL is set on the first edge of the clock when CPHA = 1 and when CPHA = 0 on the assertion of $\overline{SS}$. If TIE[1:0] bits are set to “10” then, when TUFL gets set, an interrupt will be asserted to the processor.

Note: From an application point of view, it is important to be aware that the first word available after an underflow event has occurred should be ignored, as this data was loaded into the shift register before the underflow condition was flagged.

Bit 6 = **TFE**: *Transmit FIFO Empty.*
This bit gets set whenever the Transmit FIFO has transferred its last data word to the transmit buffer. If interrupts are enabled then an interrupt will be asserted whenever the last word has been transferred to the transmit buffer.

Bit 5 = **ROFL**: *Receiver Overflow.*
This bit gets set if the Receive FIFO is full and has not been read by the processor by the time another received word arrives. If the REIE bit is set then, when this bit gets set an interrupt will be asserted to the processor. This bit is cleared when a read takes place of the CSR register and the FIFO.

Bit 4 = **RFF**: *Receive FIFO Full.*
This status bit indicates that the number of FIFO locations, as defined by the RFE[3:0] bits, are all full, i.e. if the FIFO is 4 deep then all data has been received to all four locations.
If the RIE[1:0] bits are configured as ‘11’ then, when this status bit gets set, an interrupt will be asserted to the processor. This bit is cleared when at least one data word has been read.

Bit 3 = **RFNE**: *Receive FIFO Not Empty.*
This status bit indicates that there is data in the Receive FIFO. It is set whenever there is at least one block of data in the FIFO i.e. for 8-bit mode 8 bits and for 16-bit mode 16 bits. If the RIE[1:0] bits are configured to ‘01’ then whenever this bit gets set an interrupt will be asserted to the processor. This bit is cleared when all valid data has been read out of the FIFO.

Bit 2 = **BERR**: *Bus Error.*
This status bit indicates that a Bus Error condition has occurred, i.e. that more than one

device has acted as a Master simultaneously on the BSPI bus. A Bus Error condition is defined as a condition where the Slave Select line goes active low when the module is configured as a Master, provided that MASK_SS bit in BSPIn_CSR3 register is not set. This indicates contention in that more than one node on the BSPI bus is attempting to function as a Master. This bit is cleared when the Slave Select line is deasserted, MASK_SS bit is set or Slave mode is selected.

Bit 1 = Reserved, must be kept at reset value (0).

Bit 0 = **DFIFO**: *Disable for the FIFO.*
When this bit is enabled, the FIFO pointers are all reset to 0, the RFE bits are set to 0 and therefore the BSPI is set to one location. The data within the FIFO is lost. This bit is reset to 0 after a clock cycle.

18.3.3 BSPI control/status register 3 (BSPIn_CSR3)

Address Offset: 14h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								RREQ_EN	TREQ_EN	RBURST_LEN [1:0]		TBURST_LEN [1:0]		DMA_EN	MASK_SS
-								rw	rw	rw		rw		rw	rw

This register is used to control the BSPI DMA interface. When the BSPI is receiving data, it will be the source of DMA transfer and system memory will act as destination. Conversely, when the BSPI is in transmit mode, it is sending data to an external source and it will act as DMA destination while system memory will be the source.

Bits 15:8 = Reserved, must be kept at reset value (0).

Bit 7 = **RREQ_EN**: *Receive REQuest ENable*
This is the enable bit for the reception DMA request and it flags the DMA controller that an amount of data corresponding at least to the configured reception burst length is available in the BSPI receive FIFO to be transferred.
0 = Receive DMA requests are disabled.
1 = Receive DMA requests are enabled.

Bit 6 = **TREQ_EN**: *Transmit REQuest ENable*
This is the enable bit for the transmission DMA request and it flags the DMA controller that in the BSPI transmit FIFO there is an amount of free locations corresponding at least to the configured transmission burst length.
0 = Transmit DMA requests are disabled.
1 = Transmit DMA requests are enabled.

Bits 5:4 = **RBURST_LEN[1:0]**: *Receive BURST LENgth*
These bits configure the burst length when the BSPI receives data. This programmable length is used to set the number of data words the DMA controller is expected to retrieve upon a receive request; this value is used in the DMA interface to determine when the BSPI is ready to send a data burst to the DMA. A word can be 16 bit or 8 bit wide according to the configuration set in WL bits inside BSPIn_CSR1 register.
00: 1 word transferred
01: 4 words transferred

10: 8 words transferred
11: 16 words transferred

Bits 3:2 = **TBURST_LEN [1:0]**: *Transmit BURST LENgth*
These bits configure the burst length when the BSPI transmits data. This programmable length is used to set the number of data words the DMA controller is expected to send upon a transmit request; this value is used in the DMA interface to determine when the BSPI is ready to receive a data burst from the DMA. A word can be 16 bit or 8 bit wide according to the configuration set in WL bits inside BSPIn_CSR1 register.
00: 1 word transferred
01: 4 words transferred
10: 8 words transferred
11: 16 words transferred

Bit 1 = **DMA_EN**: *DMA interface ENable*
This bit is a general enable switch for the DMA interface. When BSPI DMA interface is disabled no request line will be activated and data transfer can occur through interrupt notification only.
0 = DMA interface is disabled.
1 = DMA interface is enabled.

Bit 0 = **MASK_SS**: *MASK Slave Select*
This bit can be used to mask the status of Slave Select pin when BSPI is in master mode and the pad corresponding to SS pin is not available. When this bit is set to '1', the Bus Error interrupt condition cannot be detected since internally the related signal is always considered high regardless from the actual pad status.
0 = Slave Select pin is used.
1 = Slave Select pin is masked.

18.3.4 BSPI master clock divider register (BSPIn_CLK)

Address Offset: 10h

Reset value: 0006h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								DIV[7:0]							
-								rw							

Bits 15:8 = Reserved, must be kept at reset value (0).

Bits 7:0 = **DIV[7:0]**: *Divide factor bits.*
These bits are used to control the frequency of the BSPI serial clock with relation to the device clock. In master mode this number must be an even number greater than five, i.e. six is the lowest divide factor. In slave mode this number must be an even number greater than seven, i.e. eight is the lowest divide factor.
These bits must be set before the BSPE or MSTR bits, i.e. before the BSPI is configured into master mode.

18.3.5 BSPI transmit register (BSPIn_TXR)

Address Offset: 04h

Reset value: n/a

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16 bit Transmission	TX[15:0]															
8 bit Transmission	TX[7:0]								Not Used							
	w															

Bits 15:0 = **TX[15:0]**: *Transmit data.*
This register is used to write data for transmission into the BSPI. If the FIFO is enabled then data written to this register will be transferred to the FIFO before transmission. If the FIFO is disabled then the register contents are transferred directly to the shift register for transmission. In sixteen bit mode all of the register bits are used. In eight bit mode only the upper eight bits of the register are used while lower eight bits are ignored. In both case the data is left justified, i.e. Bit[15] = MSB, Bit[0] / Bit[8] = LSB depending on the operating mode.

18.3.6 BSPI receive register (BSPIn_RXR)

Address Offset: 00h

Reset value: 0000h

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16 bit Reception	RX[15:0]															
8 bit Reception	RX[7:0]								RX[7:0]							
	r															

Bits 15:0 = **RX[15:0]**: *Received data*.
This register contains the data received from the BSPI bus. If the FIFO is disabled then the data from the shift register is placed into the receive register directly. If the FIFO is enabled then the received data is transferred into the FIFO. In sixteen bit mode all the register bits are utilized. In eight bit reception mode the received data is replicated on the upper and lower eight bits of the register so to support both little and big endian memory systems.

18.4 BSPI register map

An overview of the BSPI registers is given in the following table.

Table 49. BSPI Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	BSPIn_RXR	RX[15:0] (1)															
04h	BSPIn_TXR	TX[15:0] (1)															
08h	BSPIn_CSR1	RFE[3:0]				WL[1:0]		CPHA	CPOL	BEIE	Reserved		REIE	RIE[1:0]		MSTR	BSPE
0Ch	BSPIn_CSR2	TIE[1:0]		TFE[3:0]				TFNE	TFF	TUFL	TFE	ROFL	RFF	RFNE	BERR	Res.	DFIFO
10h	BSPIn_CLK	Reserved								DIV[7:0]							
14h	BSPIn_CSR3	Reserved								RREQ_EN	TREQ_EN	RBURST_LEN[1:0]		TBURST_LEN [1:0]		DMA_EN	MASK_SS

1. Data is justified depending on transmission mode, Bit 15 = MSB.

See *Table 2 on page 17* for base addresses.

19 UART

19.1 Introduction

A UART interface, provides serial communication between the STR73x and other microcontrollers, microprocessors or external peripherals.

A UART supports full-duplex asynchronous communication. Eight or nine bit data transfer, parity generation, and the number of stop bits are programmable. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data can simply be double-buffered, or 16-deep fifos may be used. For multiprocessor communications, a mechanism to distinguish the address from the data bytes is included. Testing is supported by a loop-back option. A 16-bit baud rate generator provides the UART with a separate serial clock signal.

19.2 Main features

- Full-duplex asynchronous communication
- Two internal FIFOs (16 words deep) for transmit and receive data
- 16-bit baud rate generator
- Data frames both 8 and 9 bit long
- Parity bit (even or odd) and stop bit

19.3 Functional description

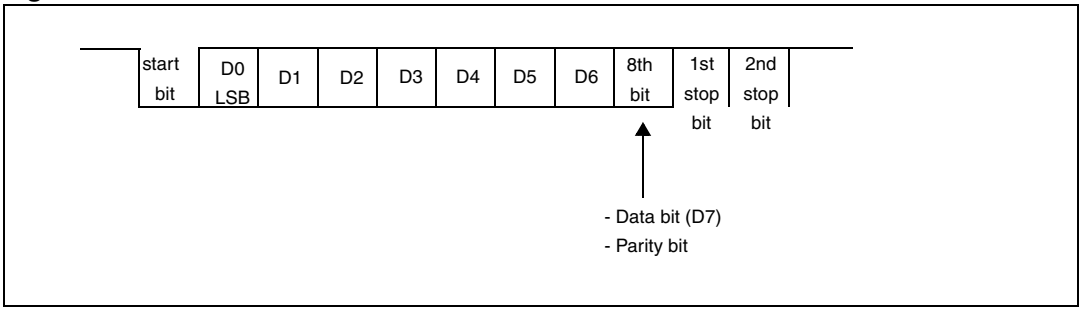
The UART supports full-duplex asynchronous communication, where both the transmitter and the receiver use the same data frame format and the same baud rate. Data is transmitted on the TXD pin and received on the RXD pin. Data frames

Eight bit data frames (see *Figure 69*) either consist of:

- eight data bits D0-7 (by setting the Mode bit field to 001);
- seven data bits D0-6 plus an automatically generated parity bit (by setting the Mode bit field to 011).

Parity may be odd or even, depending on the ParityOdd bit in the UARTn_CR register. An even parity bit will be set, if the modulo-2-sum of the seven data bits is 1. An odd parity bit will be cleared in this case.

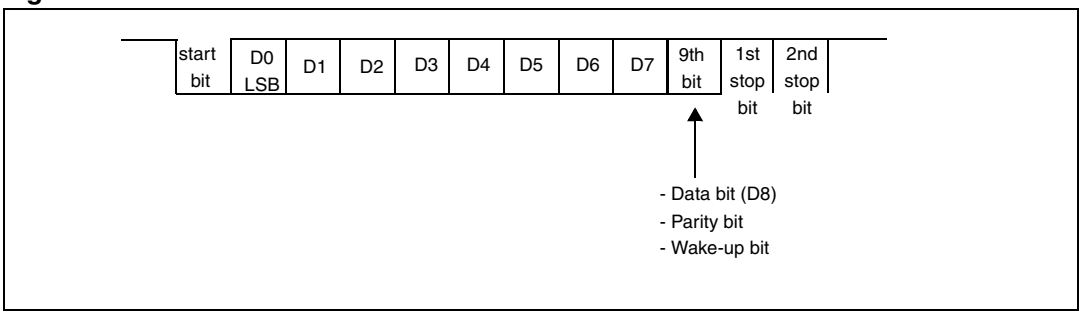
Figure 69. 8-bit data frames



Nine bit data frames (see *Figure 70*) either consist of:

- nine data bits D0-8 (by setting the Mode bit field to 100);
- eight data bits D0-7 plus an automatically generated parity bit (by setting the Mode bit field to 111);
- eight data bits D0-7 plus a wake-up bit (by setting the Mode bit field to 101).

Figure 70. 9-bit data frames



Parity may be odd or even, depending on the ParityOdd bit in the UARTn_CR register. An even parity bit will be set, if the modulo-2-sum of the eight data bits is 1. An odd parity bit will be cleared in this case.

In wake-up mode, received frames are only transferred to the receive buffer register if the ninth 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 systems. 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 ninth 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 least significant bits (LSBs) of the received character (the address). The addressed slave will switch to 9-bit data mode, which enables it to receive the data bytes that will be coming (with the wake-up bit cleared). The slaves that are not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.

19.3.1 Transmission

Values to be transmitted are written to the transmit fifo, TxFIFO, by writing to UARTn_TxBUFR. The TxFIFO is implemented as a 16 deep array of 9 bit vectors.

If the fifos are enabled (the UARTn_CR(FifoEnable) is set), the TxFIFO is considered full (UARTn_SR(TxFull) is set) when it contains 16 characters. Further writes to UARTn_TxBUFR in this situation will fail to overwrite the most recent entry in the TxFIFO. If the fifos are disabled, the TxFIFO is considered full (UARTn_SR(TxFull) is set) when it contains 1 character, and a write to UART_TxBuffer in this situation will overwrite the contents.

If the FIFOs are enabled, UARTn_SR(TxHalfEmpty) is set when the TxFIFO contains 8 or fewer characters. If the fifos are disabled, it's set when the TxFIFO is empty.

Writing anything to UARTn_TxRSTR empties the TxFIFO.

Values are shifted out of the bottom of the TxFIFO into a 9-bit txshift register in order to be transmitted. If the transmitter is idle (the txshift register is empty) and something is written to

the UARTn_TxBUFR so that the TxFIFO becomes non-empty, the txshift register is immediately loaded from the TxFIFO and transmission of the data in the txshift register begins at the next baud rate tick.

At the time the transmitter is just about to transmit the stop bits, then if the TxFIFO is non-empty, the txshift register will be immediately loaded from the TxFIFO, and transmission of this new data will begin as soon as the current stop bit period is over (i.e. the next start bit will be transmitted immediately following the current stop bit period). Thus back-to-back transmission of data can take place. If instead the TxFIFO is empty at this point, then the txshift register will become empty. UARTn_SR(TxEmpty) indicates whether the txshift register is empty.

After changing the fifoenable bit, it is important to reset the fifo to empty (by writing to the UARTn_TxRSTR register), since the state of the fifo pointer may be garbage.

The loop-back option (selected by the UARTn_CR(LoopBack) bit) internally connects the output of the transmitter shift register to the input of the receiver shift register. This may be used to test serial communication routines at an early stage without having to provide an external network.

19.3.2 Reception

Reception is initiated by a falling edge on the data input pin (RXD), provided that the UARTn_CR(Run) and UARTn_CR(RxEnable) bits are set. The RXD pin is sampled at 16 times the rate of the selected baud rate. A majority decision of the first, second and third samples of the start bit 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 falling edge transition at the RXD pin. If the start bit is valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register. For subsequent data and parity bits, the majority decision of the seventh, eighth and ninth samples in each bit time is used to determine the effective bit value.

Note: If reception is initiated when the data input pin (RXD) is being stretched at '0', a frame error is reported since the reception stage samples the initial value as a falling edge.

For 0.5 stop bits, the majority decision of the third, fourth, and fifth samples during the stop bit is used to determine the effective stop bit value.

For 1 and 2 stop bits, the majority decision of the seventh, eighth, and ninth samples during the stop bits is used to determine the effective stop bit values.

For 1.5 stop bits, the majority decision of the fifteenth, sixteenth, and seventeenth samples during the stop bits is used to determine the effective stop bit value.

The effective values received on the RXD pin are shifted into a 10-bit rxshift register.

The receive FIFO, RxFIFO, is implemented as a 16 deep array of 10-bit vectors (each 9 down to 0). If the RxFIFO is empty, UARTn_SR(RxBufNotEmpty) is set to '0'. If the RxFIFO is not empty, a read from UART_RxBFR will get the oldest entry in the RxFIFO. If fifos are disabled, the RxFIFO is considered full when it contains one character. UARTn_SR(RxHalfFull) is set when the RxFIFO contains more than 8 characters. Writing anything to UARTn_RxRSTR empties the RxFIFO.

As soon as the effective value of the last stop bit has been determined, the content of the rxshift register is transferred to the RxFIFO (except in wake-up mode, in which case this

happens only if the wake-up bit, bit8, is a ‘1’). The receive circuit then waits for the next start bit (falling edge transition) at the RXD pin.

UARTn_SR(OverrunError) is set when the RxFIFO is full and a character is loaded from the rxshift register into the RxFIFO. It is cleared when the UARTn_RxBUFR register is read.

The most significant bit of each RxFIFO entry (RxFIFO[x][9]) records whether or not there was a frame error when that entry was received (i.e. one of the effective stop bit values was ’0’). UARTn_SR(FrameError) is set when at least one of the valid entries in the RxFIFO has its MSB set.

If the mode is one where a parity bit is expected, then the bit RxFIFO[x][8] (if 8 bit data + parity mode is selected) or the bit RxFIFO[x][7] (if 7 bit data + parity mode is selected) records whether there was a parity error when that entry was received. Note, it does not contain the parity bit that was received. UARTn_SR(ParityError) is set when at least one of the valid entries in the RxFIFO has bit 8 set (if 8 bit data + parity mode is selected) or bit 7 set (if 7 bit data + parity mode is selected).

After changing the fifoenable bit, it is important to reset the fifo to empty (by writing to the UARTn_RxRSTR register), since the state of the fifo pointers may be garbage.

Reception is stopped by clearing the UARTn_CR(RxEnable) bit. A currently received frame is completed including the generation of the receive status flags. Start bits that follow this frame will not be recognized.

19.3.3 Timeout mechanism

The UART contains an 8-bit timeout counter. This reloads from UARTn_TOR whenever one or more of the following is true

- UARTn_RxBUFR is read
- The UART starts to receive a character
- UARTn_TOR is written to

If none of these conditions hold, the counter decrements towards 0 at every baud rate tick.

UARTn_SR(TimeoutNotEmpty) is ’1’ exactly whenever the RxFIFO is not empty and the timeout counter is zero.

UARTn_SR(TimeoutIdle) is ‘1’ exactly whenever the RxFIFO is empty and the timeout counter is zero.

The effect of this is that whenever the RxFIFO has got something in it, the timeout counter will decrement until something happens to the RxFIFO. If nothing happens, and the timeout counter reaches zero, the UARTn_SR(TimeoutNotEmpty) flag will be set.

When the software has emptied the RxFIFO, the timeout counter will reset and start decrementing. If no more characters arrive, when the counter reaches zero the UARTn_SR(TimeoutIdle) flag will be set.

19.3.4 Baud rate generation

The baud rate generator provides a clock at 16 times the baud rate, called the oversampling clock. This clock only ticks if UARTn_CR(Run) is set to’1’. Setting this bit to 0 will immediately freeze the state of the UART’s transmitter and receiver. This should only be done when the UART is idle.

The baud rate and the required reload value for a given baud rate can be determined by the following formulae:

Baudrate = MCLK / (16 * <UART_BaudRate>)

<UART_BaudRate> = MCLK / (16 x Baudrate)

where: <UART_BaudRate> represents the content of the UARTn_BR register, taken as unsigned 16-bit integer, and MCLK is the system clock frequency.

Table 50 and *Table 51* list various commonly used baud rates together with the required reload values and the deviation errors for two different MCLK clock frequencies (16 and 20 MHz respectively).

Table 50. Baud rates with MCLK = 16 MHz

Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
625K	1.6	2	0002	20%
38.4K	26.042	26	001A	0.160%
19.2K	52.083	52	0034	0.160%
9600	104.167	104	0068	0.160%
4800	208.333	208	00D0	0.160%
2400	416.667	417	01A1	0.080%
1200	833.333	833	0341	0.040%
600	1666.667	1667	0683	0.020%
300	3333.333	3333	0D05	0.010%
75	13333.333	13333	3415	0.003%

Table 51. Baud rates with MCLK = 20 MHz

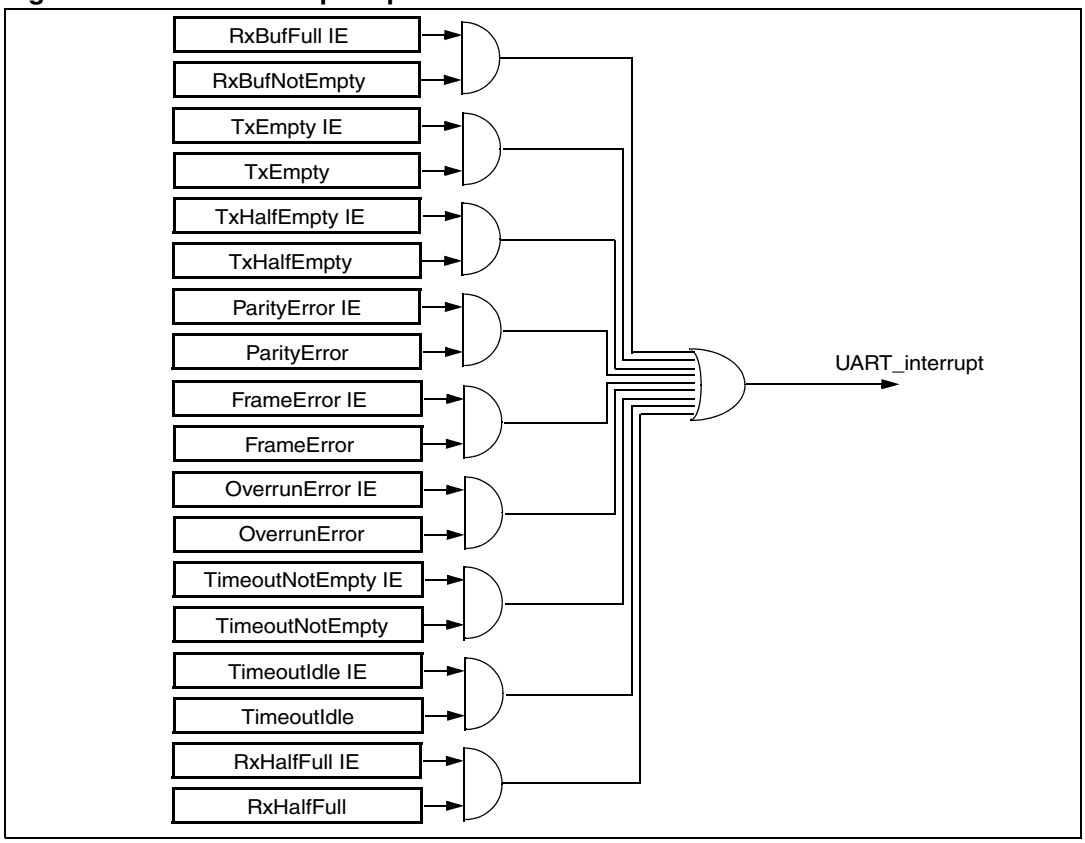
Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
625K	2	2	0002	0%
38.4K	32.552	33	0021	1.358%
19.2K	65.104	65	0041	0.160%
9600	130.208	130	0082	0.160%
4800	260.417	260	0104	0.160%
2400	520.833	521	0209	0.032%
1200	1041.667	1042	0412	0.032%
600	2083.333	2083	0823	0.016%
300	4166.667	4167	1047	0.008%
75	16666.667	16667	411B	0.002%

19.3.5 Interrupt control

The UART has a single interrupt request line, called UARTn_interrupt. The status bits in the UARTn_SR register determine the cause of the interrupt. UARTn_interrupt will go high when a status bit is 1 (high) and the corresponding bit in the UARTn_IER register is 1 (see *Figure 71*).

Note: The UARTn_SR register is read only. The Status bits can only be cleared by operating on the FIFOs. The RxFIFO and TxFIFO can > be reset by writing to the UARTn_RxReset and UARTn_TxReset registers.

Figure 71. UART interrupt request



19.3.6 Using the UART interrupts when FIFOs are disabled

When FIFOs are disabled, the UART provides three interrupt requests to control data exchange via the serial channel:

- TxHalfEmpty is activated when data is moved from UART_TxBUFR to the txshift register.
- TxEmpty is activated before the stop bit is transmitted.
- RxBufNotEmpty is activated when the received frame is moved to UART_RxBUFR.

For single transfers it is sufficient to use the transmitter interrupt (TxEmpty), which indicates that the previously loaded data has been transmitted, except for the stop bit.

For multiple back-to-back transfers using TxEmpty would leave just one stop bit time for the handler to respond to the interrupt and initiate another transmission. Using the transmit

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

TxHalfEmpty is an early trigger for the reload routine, while TxEmpty indicates the completed transmission of the data field of the frame. Therefore, software using handshake should rely on TxEmpty at the end of a data block to make sure that all data has really been transmitted.

19.3.7 Using the UART interrupts when FIFOs are enabled

To transmit a large number of characters back to back, the driver routine would write 16 characters to UART_TxBUFR, then every time a TxHalfEmpty interrupt fired, it would write 8 more. When it had nothing more to send, a TxEmpty interrupt would tell it when everything has been transmitted.

When receiving, the driver could use RxBufNotEmpty to interrupt every time a character came in. Alternatively, if data is coming in back-to-back, it could use RxHalfFull to interrupt it when there was at least 8 characters in the RxFIFO to read. It would have as long as it takes to receive 8 characters to respond to this interrupt before data would overrun. If less than eight character streamed in, and no more were received for at least a timeout period, the driver could be woken up by one of the two timeout interrupts, TimeoutNotEmpty or TimeoutIdle.

19.4 Register description

19.4.1 UART baudrate register (UARTn_BR)

Address Offset: 00h
Reset value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BaudRate[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The UARTn_BR register is the dual-function baud rate generator/reload register.

A read from this register returns the content of the timer, writing to it updates the reload register.

An auto-reload of the timer with the content of the reload register is performed each time the UARTn_BR register is written to. However, if the Run bit of the UARTn_CR register is 0 at the time the write operation to the UARTn_BR register is performed, the timer will not be reloaded until the first MCLK clock cycle after the Run bit is 1.

Bits 15:0 = **BaudRate[15:0]** *UART Baudrate*
Write function: 16-bit reload value
Read function: 16-bit count value

19.4.2 UART TxBuffer register (UARTn_TxBUFR)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							TX[8]	TX[7]	TX[6]	TX[5]	TX[4]	TX[3]	TX[2]	TX[1]	TX[0]
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Writing to the transmit buffer register starts data transmission.

Bits 15:9 = Reserved, must be kept at reset value (0).

Bit 8 = **TX[8]**: *Transmit buffer data D8.*
Transmit buffer data D8, or parity bit, or wake-up bit or undefined - dependent on the operating mode (the setting of the Mode field in UARTn_CR register).

Note: If the Mode field selects an 8 bit frame then this bit should be written as 0.

Note: If the Mode field selects a frame with parity bit, then the TX[8] bit will contain the parity bit (automatically generated by the UART). Writing '0' or '1' in this bit will have no effect on the transmitted frame.

Bit 7 = **TX[7]**: *Transmit buffer data D7*.
Transmit buffer data D7 or parity bit - dependent on the operating mode (the setting of the Mode field in UARTn_CR register).

Note: If the Mode field selects a frame with parity bit, then the TX[7] bit will contain the parity bit (automatically generated by the UART). Writing '0' or '1' in this bit will have no effect on the transmitted frame.

Bits 6:0 = **TX[6:0]**: *Transmit buffer data D(6:0)*

19.4.3 UART RxBuffer register (UARTn_RxBUFR)

Address Offset: 08h
Reset value: 0000h

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

The received data and, if provided by the selected operating mode, the received parity bit can be read from the receive buffer register.

Bits 15:10 = Reserved, always return '0' when read.

Bit 9 = **RX[9]**: *Frame error.*
If set, it indicates a frame error occurred on data stored in RX[8:0] (i.e. one of the effective stop bit values was '0' when the data was received).

Bit 8 = **RX[8]**: *Receive buffer data D8.*
Receive buffer data D8, or parity error, or wake-up bit - dependent on the operating mode (the setting of the Mode field in the UARTn_CR register).

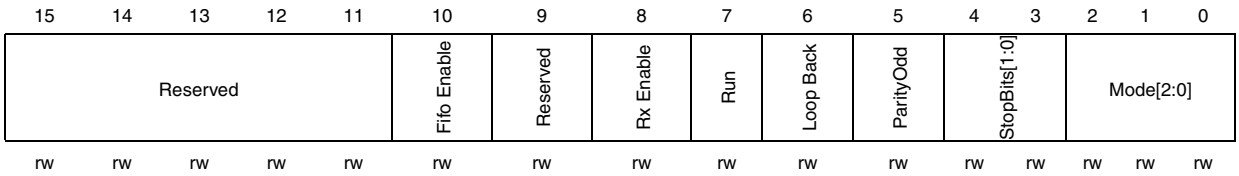
Note: If the Mode field selects a 7- or 8-bit frame then this bit is undefined. Software should ignore this bit when reading 7- or 8-bit frames.

Bit 7 = **RX[7]**: *Receive buffer data D7.*
Receive buffer data D7, or parity error - dependent on the operating mode (the setting of the Mode field in the UARTn_CR register).

Bits 6:0 = **RX[6:0]**: *Receive buffer data D(6:0).*

19.4.4 UART control register (UARTn_CR)

Address Offset: 0Ch
Reset value: 0000h



This register controls the operating mode of the UART and contains control bits for mode and error check selection, and status flags for error identification.

Note: Programming the mode control field (Mode) to one of the reserved combinations may result in unpredictable behavior.

Note: Serial data transmission or reception is only possible when the baud rate generator run bit (Run) is set to 1. When the Run bit is set to 0, TXD will be 1. Setting the Run bit to 0 will immediately freeze the state of the transmitter and receiver. This should only be done when the UART is idle.

Bits 15:11= Reserved, always return ‘0’ when read.

Bit 10 = **FifoEnable**: *FIFO Enable*
0: FIFO mode disabled
1: FIFO mode enabled

Bit 9 = Reserved
Must be kept at 0.

Bit 8 = **RxEnable**: *Receiver Enable*
0: Receiver disabled
1: Receiver enabled

Bit 7 = **Run**: *Baudrate generator Run bit*
0: Baud rate generator disabled (UART inactive)
1: Baud rate generator enabled

Bit 6 = **LoopBack**: *LoopBack mode enable*
0: Standard transmit/receive mode
1: Loopback mode enabled

Note: This bit may be modified only when the UART is inactive.

Bit 5 = **ParityOdd**: *Parity selection*
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)

Bits 4:3 = **StopBits[1:0]**: *Number of stop bits selection*
These bits select the number of stop bits
00: 0.5 stop bits
01: 1 stop bit

10: 1.5 stop bits
11: 2 stop bits

Bits 2:0 = **Mode[2:0]**: *UART Mode control*
000: reserved
001: 8 bit data
010: reserved
011: 7 bit data + parity
100: 9 bit data
101: 8 bit data + wake up bit
110: reserved
111: 8 bit data + parity

19.4.5 UART IntEnable register (UARTn_IER)

Address Offset: 10h
Reset value 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED							Rx Half FullIE	Time out IdleIE	Timeout Not Empty IE	Overrun Error IE	Frame Error IE	Parity Error IE	TxHalf Empty IE	Tx Empty IE	RxBuf NotEmptyIE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The UARTn_IE register enables the interrupt sources.

Interrupts will occur when a status bit in the UARTn_SR register is 1, and the corresponding bit in the UARTn_IER register is 1.

Bits 15:9 = Reserved, always return ‘0’ when read.

Bit 8 = **RxHalfFullIE**: *Receiver buffer Half Full Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 7 = **TimeoutIdleIE**: *Timeout Idle Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

Bit 6 = **TimeoutNotEmptyIE**: *Timeout Not Empty Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

Bit 5 = **OverrunErrorIE**: *Overrun Error Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

Bit 4 = **FrameErrorIE**: *Framing Error Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

Bit 3 = **ParityErrorIE**: *Parity Error Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

Bit 2 = **TxHalfEmptyIE**: *Transmitter buffer Half Empty Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

Bit 1 = **TxEmptyIE**: *Transmitter Empty Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

Bit 0 = **RxBufNotEmptyIE**: *Receiver Buffer Not Empty Interrupt Enable*
0: Interrupt disabled.
1: Interrupt enabled.

19.4.6 UART status register (UARTn_SR)

Address Offset: 14h
Reset value: 0006h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						TxFull	Rx HalfFull	TimeoutIdle	TimeoutNotEmpty	OverrunError	FrameError	ParityError	TxHalfEmpty	TxEmpty	RxBufNotEmpty
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

The UARTn_SR register determines the cause of an interrupt.

Bits 15:10 = Reserved, always return ‘0’ when read.

Bit 9 = **TxFull**: *TxFIFO Full.*
Set when the TxFIFO contains 16 characters.

Bit 8 = **RxHalfFull**: *RxFIFO Half Full.*
Set when the RxFIFO contains at least 8 characters.

Bit 7 = **TimeoutIdle**: *Timeout Idle*
Set when there is a timeout and the RxFIFO is empty

Bit 6 = **TimeoutNotEmpty**: TimeoutNotEmpty
Set when there is a timeout and the RxFIFO is not empty

Bit 5 = **OverrunError**: *Overrun Error*
Set when data is received and the RxFIFO is full.

Bit 4 = **FrameError**: *Frame Error*
Set when the RxFIFO contains something received with a frame error

Bit 3 = **ParityError**: *Parity Error.*
Set when the RxFIFO contains something received with a parity error

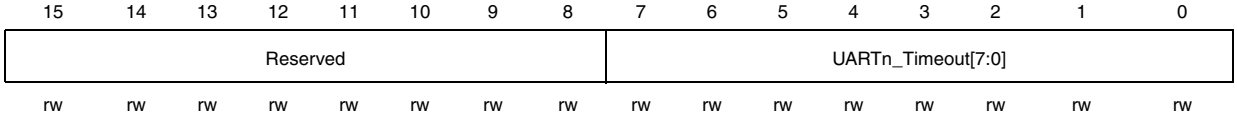
Bit 2 = **TxHalfEmpty**: *TxFIFO Half Empty.*
Set when TxFIFO at least half empty

Bit 1 = **TxEmpty**: *TxFIFO Empty.*
Set when transmit shift register is empty

Bit 0 = **RxBufNotEmpty**: *Rx Buffer Not Empty*
Set when RxFIFO not empty (RxFIFO contains at least one entry)

19.4.7 UART timeout register (UARTn_TOR)

Address Offset: 1Ch
Reset value: 0000h



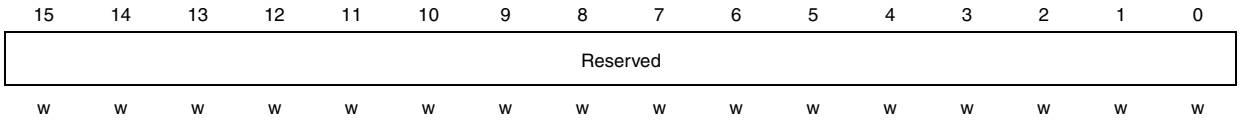
This register is to have a timeout system to be sure that not too much time pass between two successive received characters.

Bits 15:8 = Reserved, always return ‘0’ when read.

Bits 7:0 = **UARTn_Timeout[7:0]**: *Timeout.*
Timeout period in baud rate ticks.

19.4.8 UART TxReset register (UARTn_TxRSTR)

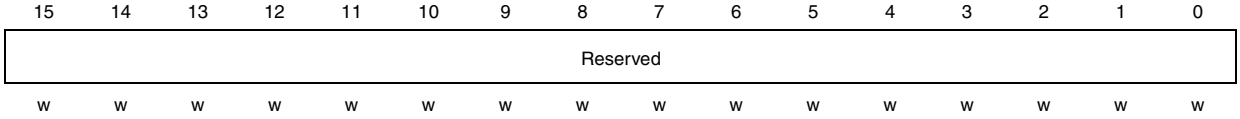
Address Offset: 20h
Reset value: Reserved



A write to this register empties the TxFIFO.

19.4.9 UART RxReset register (UARTn_RxRSTR)

Address Offset: 24h
Reset Value: Reserved



A write to this register empties the RxFIFO.

19.5 UART register map

The following table summarizes the registers implemented in the UART.

Table 52. UART register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	UARTn_BR	UART_BaudRate															
04h	UARTn_ TxBUFR	Reserved							UART_TxBuffer								
08h	UARTn_ RxBUFR	Reserved							UART_RxBuffer								
0Ch	UARTn_CR	Reserved					FifoEnable	Reserved	RxEnable	Run	LoopBack	ParityOdd	Stop Bits	Mode			
10h	UARTn_IER	Reserved							UART_IntEnable								
14h	UARTn_ SR	Reserved							UART_Status								
1Ch	UARTn_TOR	Reserved								UART_Timeout							
20h	UARTn_TxRSTR	UART_TxReset															
24h	UARTn_RxRSTR	UART_RxReset															

See *Table 2 on page 17* for the base address

20 A/D converter (ADC)

20.1 Main features

- Max Internal frequency: 10 MHz (ADC clock)
- Resolution: 10 bits
- Monotonicity: Guaranteed
- No missing codes: Guaranteed
- Zero Input reading: 0000h
- Full Scale reading: 03FFh
- Power Supply (V_{DD}): 5 V +/- 10%
- 4-bit MCLK Frequency Prescaler
- One-Shot/Scan Modes
- Chain Injection Mode
- Power Down Mode
- 16 10-bit data registers (one register for each channel)
- Four selectable analog watchdog channels
- Sampling Time: 10 ADC clock cycles
- Conversion Time: 30 ADC clock cycles (including Sampling)

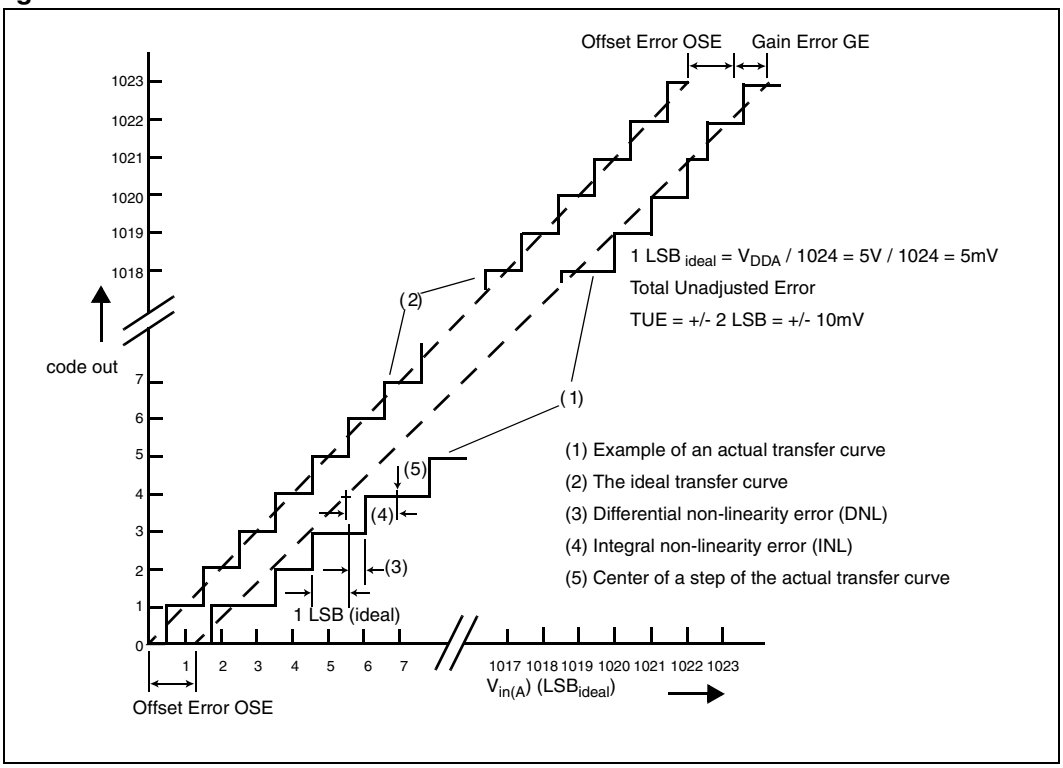
20.2 Introduction

The Analog to Digital Converter (ADC) comprises an input multiplex channel selector feeding a successive approximation converter. The conversion resolution is 10 bits.

The conversion time depends on the ADC clock frequency and the prescaler factors stored in the ADC_CLR1 register. Two prescalers can be used to provide a slow clock during sampling and a fast clock during conversion.

You can increase the conversion time by modifying the prescaling factor, for example in case you need to add an external resistor larger than 10 kW to the ADC input pin. The minimum conversion time specified in the STR73x datasheet includes the time required by the built-in Sample and Hold circuitry, which minimizes the need for external components and allows quick sampling of the signal to minimize warping and conversion error.

Figure 72. A/D conversion characteristics



The converter uses a fully differential analog input configuration for the best noise immunity and precision performance. Two separate supply references are provided to ensure the best possible supply noise rejection. In fact, the converted digital value, is referred to the analog reference voltage which determines the full scale converted value. Of course, Analog and Digital V_{SS} MUST be common (to be tied together externally). If analog supplies are not available, input reference voltages are referred to the digital ground and supply.

Up to 16 multiplexer Analog Inputs are available. A group of signals can be converted sequentially by simply programming the starting address of the first analog channel to be converted and the number of channels to convert.

An analog watchdog is provided, allowing continuous hardware monitoring of four input channels. The comparison result is stored in a dedicated register.

Single and continuous conversion modes are available and a Power-Down programmable bit allows the ADC to be set in low-power idle mode.

20.3 Functional description

20.3.1 Start of calibration

The internal calibration process can be started by setting the CAL bit in the ADC_CLR0 register, this bit remains at level 1 until the calibration is over. It is not possible to stop the calibration. If a conversion is started before the calibration is finished, the effective conversion waits the end of calibration.

The start of calibration is performed only if the PWDN bit in the ADC_CLR4 register is cleared.

20.3.2 Start of conversion

You start the programmed conversion by setting the START bit in the ADC_CLR0 register.

The start of conversion is performed only if the PWDN bit in the ADC_CLR4 register is cleared.

If you clear the START bit, the ADC will finish the current chain conversion and to clear the START bit when the operation is completed.

20.3.3 Operating modes

Two operating modes are available: One-Shot Mode and Scan Mode. You select these modes using the MODE bit in Control Logic Register 2 (ADC_CLR2).

In One-Shot Mode (MODE=0) a sequential conversion of the number of channels specified in the ADC_CLR2 register is performed once only. At the end of each conversion the digital result of the conversion is stored in the corresponding data register.

Clearing the START bit in the ADC_CLR0 register has no effect in One-Shot Mode.

In Scan Mode (MODE=1) a sequential conversion of the number of channels specified in the ADC_CLR2 register is performed continuously. At the end of each conversion the digital result of the conversion is stored in the corresponding data register. To stop Scan Mode, clear the START bit in the ADC_CLR0 register. The ADC will clear the START bit after the last conversion in the current scan is completed.

You can convert a single channel by setting the number of input channels to zero.

In both modes, at the end of each conversion an End Of Conversion interrupt request is generated (if enabled by the corresponding mask bit). At the end of the sequence, an End Of Chain interrupt request is generated (if enabled by the corresponding mask bit).

20.3.4 Input channel selection

You select the number of analog inputs to be converted by writing the number of the first channel to be converted (FCH[3:0] bits) and the total number of channels minus one (NCH[3:0] bits) in the ADC_CLR2 register. At the end of each conversion, the channel value is incremented and the number of channels to convert is decremented until the channel number is zero.

If FCH[3:0] or NCH[3:0] are programmed with a value greater than the maximum number of available channels the stored value is reset to zero.

If the sum of FCH[3:0] and NCH[3:0] is greater than the number of available channels then the conversion chain continues from channel zero after the last available channel is converted, in a circular conversion chain.

20.3.5 Analog clock prescaler

Two prescalers can be used, one defining the main ADC clock frequency (CNVP), the other (SMPP) can be optionally enabled to provide the ADC clock only during the sampling phase. This allows you to sample at low speed while converting at full speed.

The frequency of the sampling (or conversion) clock are configured independently using the SMPP[2:0] bits and CNVP[2:0] bits in the ADC_CLR1 register. The prescalers allow you to configure the ADC clock frequency based on the system clock (MCLK) with the following

division factors: 1, 2, 4, 6, 8, 10, 12 or 14. Do not exceed the max ADC clock frequency of 10 MHz when programming the prescaler.

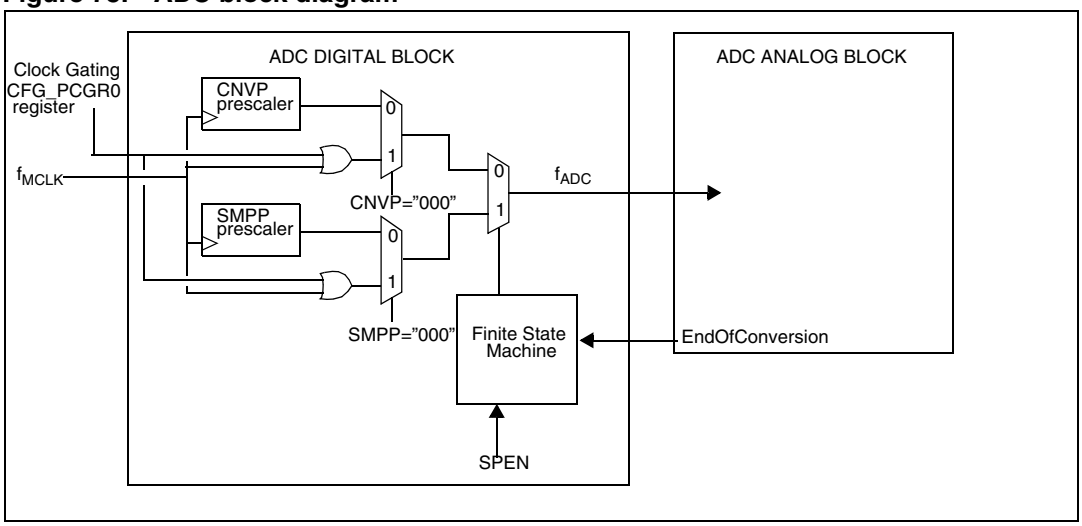
During calibration the CNVP clock is used.

You can change the prescalers only when no conversion or calibration is ongoing. To ensure this, a write to these registers becomes effective only when the conversion/calibration start bits are inactive.

Setting the SPEN bit in the ADC_CLR1 register allows you to use different clock division factors for sampling and conversion as described above. When SPEN = 0, the same clock division factor, defined by the CNVP[2:0] bits in the ADC_CLR1 register, is used for both sampling and conversion.

A state machine memorizes the state of the ADC in order to select the right clock during the sampling and the conversion phase (see *Figure 73*).

Figure 73. ADC block diagram



Note: If SPEN = 0, the sampling phase duration (tS) and the conversion phase duration (TC) are defined as stated below:

*tS = 10 * t(MCLK) * CNVP*
*tC = 20 * t(MCLK) * CNVP*

If SPEN = 1, the sampling phase duration (tS) and the conversion phase duration (TC) are defined as stated below:

*tS = (10 * t(MCLK) +1)* SMPP*
*tC = 20 * t(MCLK) * CNVP*

In both cases, the global ADC conversion time (t) is equal to:

t = tS + tC

Table 53. Prescaler Programming

CNVP[2:0] /SMMP[2:0] bits	CNVP/SMPP Value	f_{ADC}
000	1	f_{MCLK}
001	2	$f_{MCLK}/2$
010	4	$f_{MCLK}/4$
011	6	$f_{MCLK}/6$
100	8	$f_{MCLK}/8$
101	10	$f_{MCLK}/10$
110	12	$f_{MCLK}/12$
111	14	$f_{MCLK}/14$

20.3.6 Injected conversion chain

A conversion chain can be injected by setting the JSTART bit in the ADC_CLR3 register. This conversion can only be in one-shot mode and can interrupt the normal scan conversion; when the injected conversion is finished the normal conversion restarts (if pending) from the first not converted channel in the programmed chain.

At the end of each conversion an End Of Injected Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the sequence an End Of Injected Chain interrupt is issued (if enabled by the corresponding mask bit).

The first channel of the injected chain is programmed by the JFCH field in the ADC_CLR3 register. the number of converted channels is programmed by the JNCH field in ADC_CLR3 register.

When you set the JSTART bit in the ADC_CLR3 register the current conversion is terminated and the injected chain is converted. At the end of the chain the JSTART bit is reset and the normal chain conversion is resumed.

20.3.7 Analog watchdogs

Four programmable watchdogs are available for analog threshold detection.

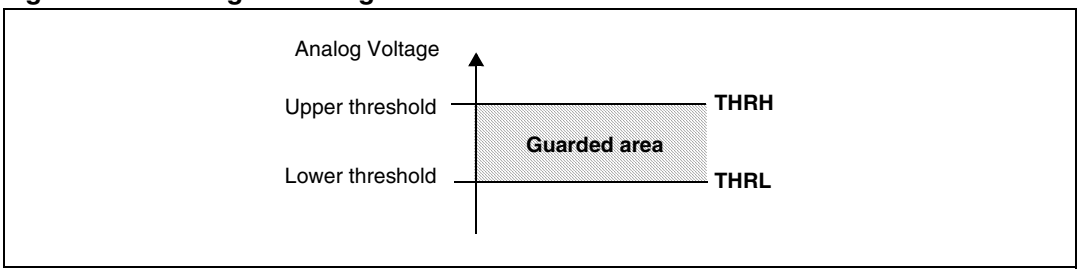
For each analog watchdog, the analog input channel to be guarded is selected by the THRCHn bits in the ADC_TRAx (x=0, 1, 2, 3) registers.

The low and high thresholds of the guarded area are selected by the THRL and THRH bits in the ADC_TRBx and ADC_TRAx (x=0, 1, 2, 3) registers.

The analog watchdogs are enabled by setting the THREN bits in the ADC_TRBx (x=0, 1, 2, 3) registers.

After the conversion of the selected channel is done, a comparison is performed between the current channel and the threshold values in THRL and THRH. The result is stored in the THRx bits in the ADC_PBR register as shown in *Table 55*, and, depending on the MSKxL and MSKxH mask bits in the ADC_IMR register, an interrupt request is generated if the converted value is outside the guarded area shown in *Figure 74*.

Figure 74. Analog watchdog thresholds



20.3.8 DMA functionality

A DMA request can be programmed after the conversion of each channel, by setting the respective masking bit DMAx (x=[0..15]) in the DMAR register. It is strongly recommended to set/reset the DMA masking bits only when the channel conversion is stopped.

The DMA transfers can only be executed if the DMAEN bit in the ADC_DMAE register is set.

When the DMAEN bit is set, the number of the first DMA-enabled conversion channel is stored in the DENCH[3:0] bits in the ADC_DMAE register.

20.3.9 Interrupts

The ADC is able to generate five interrupt requests:

- EOC (End of Conversion) interrupt request;
- ECH (End of Chain) interrupt request;
- JEOC (End of injected Conversion) interrupt request;
- JECH (End of injected Chain) interrupt request;
- Analog Watchdog interrupt requests (two pending bits for each channel)

The logical OR of all previous requests is provided to the EIC.

Two registers named ADC_PBR (Pending Bit Register) and ADC_IMR (Interrupt Mask Register) are provided in order to check and enable the interrupt request to the EIC.

The ADC_PBR register contains the interrupt pending request. The Watchdog interrupt source sets 2 pending bits (as indicated in *Table 55*) for each of the 4 guarded channels.

To reset a PBR pending bit, a register write with all bits to zero except the one to be cleared should be done by software.

This is the only write operation on the PBR register. Any other write operation is forbidden, so the register is accessible in Read/Clear mode only.

The ADC_IMR register stores the 12 bits used to enable the interrupt request sources. The register is read/write accessible.

20.3.10 Power down mode

The analog part of the ADC can be put in low power mode by setting the PWDN bit in ADC_CLR4 register.

After reset, the analog module is in power-down by default, so this state must be exited before starting any operation, by resetting the appropriate bit in ADC_CLR4.

When in power-down mode, no calibration or conversion can be started, and if a calibration or conversion is ongoing, the operation is immediately killed, and must be restarted manually (by setting the appropriate start bit) after resuming from power-down.

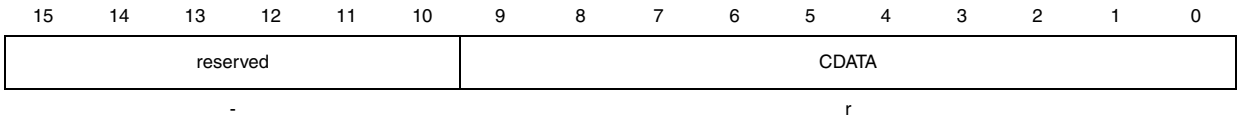
20.3.11 Auto-clock-off mode

To reduce the power consumption also during operation (without going into power-down mode) an “auto-clock-off” feature can be enabled by setting the ACKO bit in the ADC_CLR4 register. When enabled the analog clock is automatically switched off when no operation is ongoing.

20.4 Register description

20.4.1 ADC Data Register (ADC_Dx)

Address Offset: 50h...8Ch
Reset value: 0000h



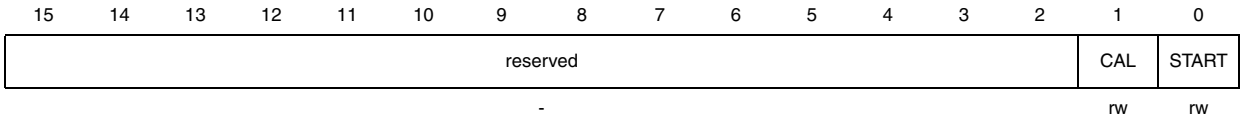
The conversion results for the 16 available channels are loaded into the 16 different Data registers following conversion of the corresponding analog input.

Bits 15:10 = Reserved, always return ‘0’ when read.

Bits 9:0 = CDATA: *Channel converted data.*

20.4.2 Control Logic Register 0 (ADC_CLR0)

Address Offset: 00h
Reset value: 0000h



Bits 15:2 = Reserved, always return ‘0’ when read.

Bit 1 = CAL: *Calibrate*.
0: No effect (write) or no calibration ongoing (read)
1: Start calibration (write) or Calibration ongoing (read). No other operation (except powerdown operation) is possible until the calibration is terminated, i.e. when this bit is cleared by hardware.

Bit 0 = START: *Start conversion*.
0: Stop conversion. Clearing this bit in Scan conversion mode causes the current chain conversion to finish and stops the operation.
1: Start chain or scan conversion. This bit stays high value while conversion is ongoing (or pending in injection mode).

20.4.3 Control Logic Register 1 (ADC_CLR1)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPEN	reserved							CNVP			reserved		SMPP		
rw	-							rw			-		rw		

Bit 15 = **SPEN**: *Sample prescaler enable.*
0: Use clock from Conversion Prescaler during sampling phase.
1: Use clock from Sampling Prescaler during sampling phase.

Bits 14:8 = Reserved, always return ‘0’ when read.

Bit 7:5 = **CNVP[2:0]**: *Conversion prescaler.*
These bits are written by software to define the prescaling factor used during calibration and conversion. Refer to *Table 54*. This clock is also used for sampling if the SPEN bit is cleared.

Bits 4:3 = Reserved, always return ‘0’ when read.

Bits 2:0 = **SMPP**: *Sampling prescaler.*
These bits are written by software to define the prescaling factor used the sampling phase of the conversion. Refer to *Table 54*. This clock is applied only during channel sampling and if the SPEN bit is set.
If a conversion or a calibration is ongoing, a write to this field becomes effective only at the end of the pending operation.

Table 54. Conversion/sampling prescaler configuration

CNVP[2:0] / SMPP[2:0]	f_{ADC}
000	MCLK
001	MCLK/2
010	MCLK/4
011	MCLK/6
100	MCLK/8
101	MCLK/10
110	MCLK/12
111	MCLK/14

20.4.4 Control logic register 2 (ADC_CLR2)

Address Offset: 08h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE	reserved					NCH				reserved		FCH			
rw	-					rw				-		rw			

Bit 15 = **MODE**: *One-Shot/Scan*.
0: One-Shot Mode: configure the conversion of one chain (from channel FCH for NCH channels).
1: Scan Mode: configure continuous chain conversion mode (from channel FCH for NCH channels); when the programmed chain conversion is finished it restarts immediately.

Bits 14:12 = Reserved, always return ‘0’ when read.

Bits 11:10 = Reserved, must be kept at reset value (0).

Bits 9:6 = **NCH[3:0]**: *Number of channels to be converted.*
These bits define the number of channels to be converted minus one.
When a channel is converted, then the number is decremented for the successive conversion, until number is zero. When conversion is ongoing a read of this register will return the number of channel to be converted before the end of chain (even in injected mode).
At the end of chain operation the value written by software is restored.

Bits 5:4 = Reserved, must be kept at reset value (0).

Bits 3:0 = **FCH[3:0]**: *First channel to be converted.*
These bits define the starting analog input channel.
The first channel addressed by FCH is converted, then the channel number is incremented for the successive conversion, until last channel is converted. When conversion is ongoing a read of this register will return the number of the currently converted channel (or the pending channel during injected mode).
When the maximum available channel is reached and NCH is not zero, the next converted channel is set to zero.
At the end of chain operation the value written by software is restored.

20.4.5 Control logic register 3 (ADC_CLR3)

Address Offset: 0Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSTART	reserved					JNCH				reserved		JFCH			
rw	-					rw				-		rw			

Bit 15 = **JSTART**: *Injection start*.
0: No effect (injected conversion cannot be interrupted)
1: Start conversion of injected analog channels

Bits 14:10 = Reserved, must be kept at reset value (0).

Bits 9:6 = **JNCH[2:0]**: *Number of Injected channels to convert.*
These bits define the number of analog input channels minus one to be used for channel injection.

Bits 5:4 = Reserved, must be kept at reset value (0).

Bits 3:0 = **JFCH[3:0]**: *First Injected channel to convert.*
These bits define the starting analog input channel for the injected conversion.

20.4.6 Control logic register 4 (ADC_CLR4)

Address Offset: 10h
Reset value: 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PWDN	ACKO	reserved								NOAVRG	reserved				
rw	rw	-								rw	-				

Bit 15 = **PWDN**: *Power down enable*.
0: Powerdown mode disabled (no effect if a conversion or calibration is ongoing)
1: Powerdown mode enabled

Bit 14 = **ACKO**: *Auto clock off enable.*
0: Auto clock off feature disabled
1: Auto clock off feature enabled (switch off analog clock while not converting or calibrating).

Bits 13:6 = Reserved, always return ‘0’ when read.

Bit 5 = **NOAVRG**: *No calibration average enable.*
This bit is set and cleared by software.
0: Noise filtering enabled during calibration
1: Noise filtering disabled during calibration (cuts calibration time by 16).

Bits 4:0 = Reserved, must be kept at reset value (0).

20.4.7 Threshold registers A (ADC_TRA0 ..3)

Address Offset: 14h, 18h, 1Ch, 20h
Reset value: 0000h

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

The four TRAx (x = 0,1, 2, 3) registers are used to store the 10-bit user-programmable upper threshold values and the channels linked to the threshold detection.

Bits 15:14 = Reserved, must be kept at reset value (0).

Bits 13:10 = **THRCH**: *Channel linked to threshold detection.*
These bits define the analog input channel to be used for threshold detection channel x (x = 0,1, 2, 3).

Bits 9:0 = **THRH**: *High threshold value for channel x.*

20.4.8 Threshold registers B (ADC_TRB0 ..3)

Address Offset: 24h, 28h, 2Ch, 30h
Reset value: 0000h

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

The four TRBx (x = 0,1, 2, 3) registers are used to store the 10-bit user-programmable lower threshold values and to enable/disable the threshold detection.

Bit 15 = **THREN**: *Threshold enable.*
When set enables the threshold detection feature for the selected channel.

Bits 14:10 = Reserved, must be kept at reset value (0).

Bits 9:0 = **THRL**: *Low threshold value for channel x*.

20.4.9 DMA channel enable register (ADC_DMAR)

Address Offset: 34h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA15	DMA14	DMA13	DMA12	DMA11	DMA10	DMA9	DMA8	DMA7	DMA6	DMA5	DMA4	DMA3	DMA2	DMA1	DMA0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 = **DMA[15:0]**: *DMA enable for channel n.*
0: DMA capability is disabled for channel n.
1: Channel n is enabled to transfer data in DMA mode

20.4.10 DMA global enable register (ADC_DMAE)

Address Offset: 44h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAEN	reserved											DENCH			
rw	-											r			

Bit 15 = **DMAEN**: *DMA global enable.*

This bit must be set by software to globally enable the DMA feature.
0: DMA disabled
1: DMA enabled

Bits 14:4 = Reserved, must be kept at reset value (0).

Bits 3:0 = **DENCH**: *DMA first enabled channel.*
The number of the first DMA-enabled channel valid when DMAEN bit was last set (read only).

20.4.11 Pending bit register (ADC_PBR)

Address Offset: 48h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				THR3		THR2		THR1		THR0		JEOC	JECH	EOC	ECH
-				rc		rc		rc		rc		rc	rc	rc	rc

Bits 15:12 = Reserved, always return ‘0’ when read.

Bits 11:10 = **THR3[1:0]**: *Threshold Detection channel 3.*
These bits contain the result of the comparison on Analog Watchdog Threshold Channel 3. Refer to *Table 55*. Comparison takes place only if the THREN bit in the ADC_TRA3 register is set. Threshold channel 3 is associated with an analog input channel using the THRCH bits in the ADC_TRA3 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register

Table 55. Meaning of Analog Watchdog THRx[1:0] bits

THRx[1:0] bits	Meaning
10	converted data >= THRH
01	converted data < THRL
00	THRL <= converted data < THRH

Bits 9:8 = **THR2[1:0]**: *Threshold Detection channel 2.*
These bits contain the result of the comparison on Analog Watchdog Threshold Channel 2. Refer to *Table 55*. Comparison takes place only if the THREN bit in the ADC_TRA2 register is set. Threshold channel 2 is associated with an analog input channel using the THRCH bits in the ADC_TRA2 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register

Bits 7:6 = **THR1[1:0]**: *Threshold Detection channel 1.*
These bits contain the result of the comparison on Analog Watchdog Threshold Channel 1. Refer to *Table 55*. Comparison takes place only if the THREN bit in the ADC_TRA1 register is set. Threshold channel 1 is associated with an analog input channel using the THRCH bits in the ADC_TRA1 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register

Bits 5:4 = **THR0[1:0]**: *Threshold Detection channel 0.*
These bits contain the result of the comparison on Analog Watchdog Threshold Channel 0. Refer to *Table 55*. Comparison takes place only if the THREN bit in the ADC_TRA0 register is set. Threshold channel 0 is associated with an analog input channel using the THRCH bits in the ADC_TRA0 register. An interrupt request is generated if enabled by the corresponding MSK bit in the ADC_IMR register

Bit 3 = **JEOC**: *End of injected channel conversion.*
This bit is set by hardware at the End Of Injected Conversion started by the JSTART bit. An interrupt request is generated if enabled by the MSKJEOC bit in the ADC_IMR register.
0: No JEOC event
1: End of conversion of injected channel

Bit 2 = **JECH**: *End of injected chain conversion.*
This bit is set by hardware at the End of Injected Chain conversion started by the JSTART bit. An interrupt request is generated if enabled by the MSKJECH bit in the ADC_IMR register.
0: No JECH event
1: End of injected Chain Conversion

Bit 1 = **EOC**: *End of conversion.*
This bit is set by hardware at the End of Conversion started by the START bit. An interrupt request is generated if enabled by the MSKEOCH bit in the ADC_IMR register.
0: No EOC event
1: End Conversion

Bit 0 = **ECH**: *End of chain conversion.*
This bit is set by hardware at the End of Chain conversion started by the START bit. An interrupt request is generated if enabled by the MSKECH bit in the ADC_IMR register.
0: No ECH event
1: End of Chain Conversion

20.4.12 Interrupt mask register (ADC_IMR)

Address Offset: 4Ch
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				MSK3H	MSK3L	MSK2H	MSK2L	MSK1H	MSK1L	MSK0H	MSK0L	MSK JEOC	MSK JECH	MSK EOC	MSK ECH
-				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:12 = Reserved, always return ‘0’ when read.

Bit 11 = **MSK3H**: *Analog Watchdog 3 High Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR3H interrupt request disabled
1: THR3H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 3 is greater than or equal to the high threshold (THR3[1:0]=10).

Bit 10 = **MSK3L**: *Analog Watchdog 3 Low Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR3L interrupt request disabled
1: THR3L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 3 is less than the low threshold (THR3[1:0]=01).

Bit 9 = **MSK2H**: *Analog Watchdog 2 High Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR2H interrupt request disabled
1: THR2H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 2 is greater than or equal to the high threshold (THR2[1:0]=10).

Bit 8 = **MSK2L**: *Analog Watchdog 2 Low Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR2L interrupt request disabled

1: THR2L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 2 is less than the low threshold (THR1[1:0]=01).

Bit 7 = **MSK1H**:*Analog Watchdog 1 High Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR1H interrupt request disabled
1: THR1H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 1 is greater than or equal to the high threshold (THR1[1:0]=10).

Bit 6 = **MSK1L**: *Analog Watchdog 1 Low Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR1L interrupt request disabled
1: THR1L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 1 is less than the low threshold (THR1[1:0]=01).

Bit 5 = **MSK0H**: *Analog Watchdog 0 High Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR0H interrupt request disabled
1: THR0H interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 0 is greater than or equal to the high threshold (THR0[1:0]=10).

Bit 4 = **MSK0L**: *Analog Watchdog 0 Low Threshold Interrupt Enable.*
This bit is set and cleared by software.
0: THR0L interrupt request disabled
1: THR0L interrupt request enabled. An interrupt request is generated if the voltage on the channel guarded by Analog Watchdog 0 is less than the low threshold (THR0[1:0]=01)

Bit 3 = **MSKJEOC**: *Injected End of Conversion Interrupt Enable.*
This bit is set and cleared by software.
0: JEOC interrupt request disabled
1: JEOC interrupt request enabled. The JEOC bit in the ADC_PBR register is set when an interrupt request is generated.

Bit 2 = **MSKJECH**: *Injected End of Chain Conversion Interrupt Enable.*
This bit is set and cleared by software.
0: JECH interrupt request disabled
1: JECH interrupt request enabled. The JECH bit in the ADC_PBR register is set when an interrupt request is generated.

Bit 1 = **MSKEOC**: *End of Conversion Interrupt Enable.*
This bit is set and cleared by software.
0: EOC interrupt request disabled
1: EOC interrupt request enabled. The EOC bit in the ADC_PBR register is set when an interrupt request is generated.

Bit 0 = **MSKECH**: *End of Chain Conversion Interrupt Enable.*
This bit is set and cleared by software.
0: ECH interrupt request disabled
1: ECH interrupt request enabled. The ECH bit in the ADC_PBR register is set when an interrupt request is generated.

20.5 ADC register map

Table 56. ADC register map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	ADC_CLR0	reserved														CAL	START
04h	ADC_CLR1	SPEN	reserved							CNVP		reserved		SMPP			
08h	ADC_CLR2	MODE	reserved					NCH				reserved		FCH			
0Ch	ADC_CLR3	J START	reserved					JNCH				reserved		JFCH			
10h	ADC_CLR4	PWDN	ACK O	reserved								NO AVR G	reserved				
14h	ADC_TRA0	THRCH						THRH									
18h	ADC_TRA1	THRCH						THRH									
1Ch	ADC_TRA2	THRCH						THRH									
20h	ADC_TRA3	THRCH						THRH									
24h	ADC_TRB0	THR EN	reserved					THRL									
28h	ADC_TRB1	THR EN	reserved					THRL									
2Ch	ADC_TRB2	THR EN	reserved					THRL									
30h	ADC_TRB3	THR EN	reserved					THRL									
34h	ADC_DMAR	DMA 15	DMA 14	DMA 13	DMA 12	DMA 11	DMA 10	DMA 9	DMA 8	DMA 7	DMA 6	DMA 5	DMA 4	DMA 3	DMA 2	DM A1	DMA0
44h	ADC_DMAE	DMAE N	reserved											DENCH			
48h	ADC_PBR	reserved				THR3		THR2		THR1		THR0		JEO C	JEC H	EOC	ECH
4Ch	ADC_IMR	reserved				MSK 3H	MSK 3L	MSK 2H	MSK 2L	MSK 1H	MSK 1L	MSK 0H	MSK 0L	MSK JEO C	MSK JEC H	MSK EOC	MSKE CH
50h	ADC_D0	reserved					CDATA										
...	...	...					...										
8Ch	ADC_D15	reserved					CDATA										

21 APB bridge

The APB Bridge allows the connection between the ARM7TDMI native bus and the devices available on the Advanced Peripheral Bus (APB) rev. E bus. Two separate bridges are implemented on STR73x, each one requiring an ARM memory window of 16KB.

The APB memory space decoding is provided by the APB Bridge. Peripherals mapped on APB bus are aligned to word boundaries. The two APB bridges subdivide the APB space in 32 sub-pages of 1KBytes each, dedicated to the different STR73x APB mapped peripherals, and to APB Bridge Registers.

21.1 Register description

Each Bridge module implements four internal registers, 32 bits wide, mapped in the APB BRIDGE REGISTER window. These registers are used for error detection and status report.

Note: The APB Bridge registers MUST be accessed with 32 bit aligned operations (i.e. no byte/half word cycles are allowed).

Bridge Status Register (APBn_BSR)

Address Offset: 00h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved										APBT	OUTM	reserved			ABORT
-										rc	rc	-			rc

This is a READ/CLEAR register, that means that it can be read and a write with ‘0’ has no effect, while a write with ‘1’ clears the bit (reset to ‘0’).

Bits 31:6 = *Reserved*.

Bit 5 = **APBT**: *APB Time-out Flag.*
It is set when an APB Time-out condition occurs.

Bit 4 = **OUTM**: *Out of Memory.*
It is set when an Out of Memory condition occurs: this error generates an abort termination (if enabled) on the ARM7TDMI bus.

When the APB Bridge is selected, but the address value points to the reserved memory window dedicated to the Bridge registers but outside the first 16 locations (that is with address in the range of x010h to x3FFh), an out of memory condition is detected. This condition can be generated by a prohibited access to the address space of the APB Bridge internal register only. On STR73x implementation no exceptions are generated when sofware accesses reserved address windows dedicated to the peripherals.

Bits 3:1 = Reserved, must be kept at reset value.

Bit 0 = **ABORT**: *Abort Flag.*
It is set when an ARM7TDMI memory cycle has been terminated with the abort condition by the Bridge (the source of the abort termination can be read in bit 4 and 5 of the same register). In case of Time-out, the ABORT is generated ONLY for read accesses to the peripherals. The write access, due to the fact that it is posted, cannot report the APB Time-out condition with an ABORT termination (the operation is no more present on the ARM7TDMI bus).

Time-out Register (APBn_TOR)

Address Offset: 04h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
-															

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

Bits 31:9 = Reserved, must be kept at reset value.

Bit 8 = **ABTEN**: *Abort Enable.*
When set to '1' it enables the abort generation, on the ARM7TDMI bus, when an APB Time-out or an out-of-memory condition occur. If not enabled, in case of error, the Bridge will set the APBT bit in the Status Register, but will normally terminate the operation on the ARM bus.

Bits 7:5 = Reserved, must be kept at reset value.

Bits 4:0 = **TOUT_CNT**: *Time-out Counter.*
When "00000" the time-out counter is disabled; when different from zero, it represents the delay, in term of APB clock periods, the Bridge can wait for a target completion, before asserting the time-out error flag.

Out of Memory Register (APBn_OMR)

Address Offset: 08h
Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved								Peripheral Address (15:8)							
-								r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Peripheral Address (7:0)								nRW	reserved						
r								r	-						

Bits 31:24 = Reserved, must be kept at reset value.

Bits 23:8 = *Peripheral Address [15:0].*
The Bridge is able to detect an out of memory condition on the APB bus, when the ARM addresses the reserved memory window (address in the range of x010h to x3FFh in APB register window only). If this condition occurs, the Bridge ends the APB transaction and reports on the ARM7TDMI bus an ABORT condition (if enabled). The address of slave generating the error condition is reported in the Peripheral Address field.

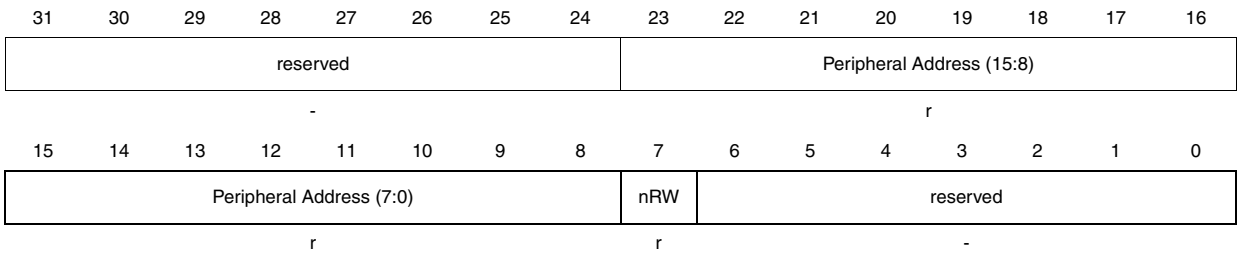
Bit 7 = **nRW**: *Read/Write Operation Flag.*
It indicates the type of operation (Read='0' or Write='1') that generated the out of memory error condition.

Bits 6:0 = Reserved, must be kept at reset value.

Note: This register is updated only at the first Out of Memory condition occurrence: it will contain information about a new one only if OUTM flag (bit 4 of Status Register) will have been cleared by the CPU.

Time-out Error Register (APBn_TOER)

Address Offset: 0Ch
Reset value: 0000 0000h



Bits 31:24 = Reserved, must be kept at reset value.

Bits 23:8 = *Peripheral Address (15:0).*
The Bridge is able to detect a time-out condition on the APB bus, when a target device takes too much time in responding to the ARM, in particular longer than the defined TOUT_CNT. If this condition occurs, the Bridge ends the APB transaction and reports on the ARM7TDMI bus an ABORT condition (if enabled). The address of slave generating the error condition is reported in the Peripheral Address field.

For further information on abort mode, please refer to ARM7TDMI datasheet.

Bit 7 = **nRW**: *Read/Write Operation Flag.*
It indicates the type of operation (Read='0' or Write='1') that generated the time-out condition.

Bits 6:0 = *Reserved*.

Note: This register is updated only at the first time-out condition: it will contain information about a new one only after the APBT flag (bit 5 of Status Register) will have been cleared by the CPU.

21.2 APB register map

Table 57. APB register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	APBn_BSR	reserved																										ABPT	OUTM	reserved			ABORT
04h	APBn_TOR	reserved																							ABTEN	reserved			TOUT_CNT				
08h	APBn_OMR	reserved								Peripheral Address																nRW	reserved							
0Ch	APBn_TOER	reserved								Peripheral Address																nRW	reserved							

See *Table 2* for base addresses

22 JTAG interface

The ARM7TDMI implements the so called Debug Interface, based on IEEE Std. 1149.1-1993, “Standard Test Access Port-Scan Boundary Architecture”. For concept and terms used in this section refer first of all to this standard; secondly, since ARM7TDMI contains hardware extensions of the standard for advanced debugging features, refer also to the document “ARM7TDMI Data Sheet” Ref. ARM DDI 0029E.

22.1 Pins and reset status

The IEEE standard gives a set of indications about how to implement internal resistors on JTAG pins; they are briefly summarized below.

$\overline{\text{JTRST}}$: To be held low at Power-on in such a way to produce an initialization (reset) of the module. When out of reset the pin shall be Pulled up. When JTAG is not in use, it may be anyway held under reset status, grounding permanently $\overline{\text{JTRST}}$ pin.

JTDI: To be Pulled up by external resistor.

JTMS: To be Pulled up by external resistor. In particular it must be high during a ‘0’ to ‘1’ transition of $\overline{\text{JTRST}}$.

JTCK: The JTAG finite state machine shall maintain the state indefinitely when JTCK is held low; optionally, a similar behavior is obtained holding high the pin JTCK as well. The standard does not impose any pull-up or pull-down device. Anyway, a floating input is not recommended to avoid static power consumption.

JTDO: High impedance when not in use.

According to the standard, in STR73x system device the following internal resistor are required, hence they need to be implemented on the host board.

- $\overline{\text{JTRST}}$ Pull-up
- JTDIPull-up
- JTMSPull-up
- JTCK Pull-down
- JTDO Floating or Pull-up/down (no static consumption anyway)

If the JTAG interface is never used, all the pins can be grounded indefinitely.

22.1.1 JTAG ID code

STR73x devices have two JTAG ID codes:

- JTAG standard ID code (accessible with standard JTAG instruction = 0b 1110):
  - 0x3F0F0F0F
- Device ID code (accessible with extended JTAG instruction = 0b 0001):
  - 0xv224F041 (where v bits are reserved and to be ignored)

23 Revision history

Table 58. Document revision history

Date	Revision	Changes
20-Sep-2005	1	Initial release.
10-Sep-2007	2	Updated *Section 14.3.10: Pulse width modulation mode on page 166* Updated *Table 7: Operating modes using main crystal controlled oscillator on page 45* Modified *Section 6.1.2: Alternate function I/O (AF) on page 68* Added *Section 11.3.3: RTC flag assertion on page 138* Modified *Section 11.4.3: RTC Prescaler Load Register High (RTC_PRLH)* and *Section 11.4.4: RTC Prescaler Load Register Low (RTC_PRLL) on page 143* Updated BSPI *Section 18.2.5: Start-up status on page 247* Updated *Section 19.4.4: UART control register (UARTn_CR) on page 264* Updated *Figure 62: Event flags and interrupt generation on page 233* Added *Section 22.1.1: JTAG ID code on page 288*
11-Jun-2008	3	Updated *Section 18.3.4: BSPI master clock divider register (BSPIn_CLK) on page 254*.

Index

A

ADC_CLR0 275
ADC_CLR1 276
ADC_Dx 275
APBn_BSR 284
APBn_OMR 285
APBn_TOER 286
APBn_TOR 285
ARB_CTLR 130
ARB_PRIOR 129
ARB_TOR 129

B

BSPIn_CLK 254
BSPIn_CSR1 248
BSPIn_CSR2 250
BSPIn_CSR3 252
BSPIn_RXR 255

C

CAN_BRPR 194
CAN_BTR 193
CAN_CR 190
CAN_ERR 193
CAN_IDR 204
CAN_IFn_A1R 199
CAN_IFn_A2R 200
CAN_IFn_CMR 197
CAN_IFn_CRR 196
CAN_IFn_DAnR 200
CAN_IFn_DBnR 200
CAN_IFn_M1R 199
CAN_IFn_M2R 199
CAN_IFn_MCR 200
CAN_IPnR 206
CAN_MVnR 207
CAN_NDnR 206
CAN_SR 191
CAN_TESTR 194
CAN_TxRnR 205
CFG_DIDR 55
CFG_EITE0 55
CFG_EITE1 56
CFG_EITE2 56
CFG_ESPR 63
CFG_PCGR0 57
CFG_PCGR1 59
CFG_PCGRB0 60
CFG_PCGRB1 61
CFG_PECGR0 62
CFG_PECGR1 62
CFG_R0 53
CFG_R1 54
CFG_TIMSR 61
CMU_CTRL 37
CMU_EOCV 40
CMU_FDISP 36
CMU_FRH 36
CMU_FRL 36
CMU_IM 40
CMU_IS 39
CMU_RCCTL 35
CMU_STAT 38
CMU_WE 41

D

DMAn_CLR 124
DMAn_CTRL3 119
DMAn_CTRLx 118
DMAn_DECURRHx 121
DMAn_DECURRLx 121
DMAn_DESTHx 117
DMAn_DESTLx 117
DMAn_MASK 123
DMAn_MAXx 117
DMAn_SOCURRHx 120
DMAn_SOCURRLx 121
DMAn_SOURCEHx 117
DMAn_SOURCELx 116
DMAn_STATUS 125
DMAn_TCNTx 122

E

EIC_CICR 90
EIC_CIPR 91
EIC_FIER 92
EIC_FIPR 92
EIC_FIR 94
EIC_ICR 89
EIC_IER0 94
EIC_IER1 95
EIC_IPR0 96
EIC_IPR1 97

EIC_IVR 93
EIC_SIRn 98

I

I/O Port Register
PC0 72
PC1 72
PC2 72
PD 73
I2C_CCR 238
I2C_CR 234
I2C_DR 240
I2C_ECCR 239
I2C_OAR1 239
I2C_OAR2 239
I2C_SR1 235
I2C_SR2 237

P

PRCCU_ CCR 45
PRCCU_CFR 47
PRCCU_PLLCR 49
PRCCU_RTCPR 51
PRCCU_SMR 50
PRCCU_VRCTR 46
PWMn_CPI 181
PWMn_DUT 182
PWMn_IM 181
PWMn_PEN 180
PWMn_PER 182
PWMn_PLS 181
PWMn_PRS0 180
PWMn_PRS1 180

R

RTC_ALRH 145
RTC_ALRL 145
RTC_CNTH 144
RTC_CNTL 144
RTC_CRH 140
RTC_CRL 141
RTC_DIVH 143
RTC_DIVL 144
RTC_PRLH 143
RTC_PRLL 143

T

TBn_CNT 155
TBn_CR 154
TBn_MR 156
TBn_PR 155
TBn_SR 156
TBn_VR 155
TIMn_CNTR 172
TIMn_CR1 173
TIMn_CR2 174
TIMn_ICAR 171
TIMn_ICBR 172
TIMn_OCAR 172
TIMn_OCBR 172
TIMn_SR 175

U

UART_BR 262
UART_CR 264
UART_IER 265
UART_RxBUFR 263
UART_SR 266
UART_TOR 267
UART_TxBUFR 263
UART_TxRSTR 267

W

WDG_CNT 150
WDG_CR 149
WDG_KR 151
WDG_MR 151
WDG_PR 149
WDG_SR 151
WDG_VR 149
WIU_CTRL 108
WIU_INTR 110
WIU_MR 109
WIU_PR 111
WIU_TR 110
WUT_CNT 134
WUT_CR 133
WUT_MR 135
WUT_PR 133
WUT_SR 134
WUT_VR 134

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.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

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

www.st.com