

---

**Support microcontrollers SPC560B54x, SPC560B60x and SPC560B64x**

---

## Introduction

The SPC560B54/6x is a family of Power Architecture<sup>®</sup>-based microcontrollers that target automotive vehicle body applications such as:

- Central body electronics
- Vehicle body controllers
- Smart junction boxes
- Front modules
- Body peripherals
- Door control
- Seat control

The SPC560B54/6x family expands the range of the SPC560B microcontroller family. It provides the scalability needed to implement platform approaches and delivers the performance required by increasingly sophisticated software architectures. The advanced and cost-efficient host processor core of the SPC560B54/6x automotive controller family complies with the Power Architecture embedded category, and only implements the VLE (variable-length encoding) APU, providing improved code density. It operates at speeds of up to 64 MHz and offers high performance processing optimized for low power consumption. It also capitalizes on the available development infrastructure of current Power Architecture devices and is supported with software drivers, operating systems and configuration code to assist with users implementations.

This document describes the features of the family and options available within the family members, and highlights important electrical and physical characteristics of the device.

# Contents

<b>1</b>	<b>Preface</b> .....	<b>47</b>
1.1	Overview .....	47
1.2	Audience .....	47
1.3	Guide to this reference manual .....	47
1.4	Register description conventions .....	51
1.5	References .....	52
1.6	How to use the SPC560B54/6x documents .....	52
1.6.1	The SPC560B54/6x document set .....	52
1.6.2	Reference manual content .....	53
1.7	Using the SPC560B54/6x .....	54
1.7.1	Hardware design .....	55
1.7.2	Input/output pins .....	55
1.7.3	Software design .....	56
1.7.4	Other features .....	56
<b>2</b>	<b>Introduction</b> .....	<b>58</b>
2.1	The SPC560B54/6x microcontroller family .....	58
2.2	SPC560B54/6x device comparison .....	58
2.3	Device block diagram .....	59
2.4	Feature details .....	61
2.4.1	e200z0h core processor .....	61
2.4.2	Crossbar switch (XBAR) .....	61
2.4.3	Interrupt Controller (INTC) .....	62
2.4.4	System Integration Unit Lite (SIUL) .....	62
2.4.5	Flash memory .....	63
2.4.6	SRAM .....	63
2.4.7	Memory Protection Unit (MPU) .....	64
2.4.8	Boot Assist Module (BAM) .....	64
2.4.9	Enhanced Modular Input Output System (eMIOS) .....	64
2.4.10	Deserial Serial Peripheral Interface Module (DSPI) .....	66
2.4.11	Controller Area Network module (FlexCAN) .....	67
2.4.12	System clocks and clock generation .....	68
2.4.13	System timers .....	68

2.4.14	System watchdog timer	69
2.4.15	Inter-Integrated Circuit (I2C) module	70
2.4.16	On-chip voltage regulator (VREG)	70
2.4.17	Analog-to-Digital Converter (ADC)	71
2.4.18	Enhanced Direct Memory Access controller (eDMA)	72
2.4.19	Cross Trigger Unit (CTU)	72
2.4.20	Serial communication interface module (LINFlex)	73
2.4.21	Nexus Port Controller (NPC)	74
2.4.22	JTAG Controller (JTAGC)	74
2.5	Developer support	74
<b>3</b>	<b>Memory Map</b>	<b>76</b>
<b>4</b>	<b>Signal description</b>	<b>79</b>
4.1	Package pinouts	79
4.2	Pad configuration during reset phases	82
4.3	Pad configuration during standby mode exit	83
4.4	Voltage supply pins	83
4.5	Pad types	84
4.6	System pins	85
4.7	Functional port pins	85
4.8	Nexus 2+ pins	109
<b>5</b>	<b>Microcontroller Boot</b>	<b>111</b>
5.1	Boot mechanism	111
5.1.1	Flash memory boot	112
5.1.2	Serial boot mode	114
5.1.3	Censorship	115
5.2	Boot Assist Module (BAM)	119
5.2.1	BAM software flow	119
5.2.2	LINFlex (RS232) boot	127
5.2.3	FlexCAN boot	128
5.3	System Status and Configuration Module (SSCM)	130
5.3.1	Introduction	130
5.3.2	Features	130
5.3.3	Modes of operation	131

5.3.4 Memory map and register description ..... 131

**6 Clock Description ..... 138**

6.1 Clock architecture ..... 138

6.2 Clock gating ..... 139

6.3 Fast external crystal oscillator (FXOSC) digital interface ..... 140

6.3.1 Main features ..... 140

6.3.2 Functional description ..... 140

6.3.3 Register description ..... 142

6.4 Slow external crystal oscillator (SXOSC) digital interface ..... 143

6.4.1 Introduction ..... 143

6.4.2 Main features ..... 143

6.4.3 Functional description ..... 143

6.4.4 Register description ..... 144

6.5 Slow internal RC oscillator (SIRC) digital interface ..... 145

6.5.1 Introduction ..... 145

6.5.2 Functional description ..... 145

6.5.3 Register description ..... 146

6.6 Fast internal RC oscillator (FIRC) digital interface ..... 146

6.6.1 Introduction ..... 146

6.6.2 Functional description ..... 147

6.6.3 Register description ..... 148

6.7 Frequency-modulated phase-locked loop (FMPLL) ..... 148

6.7.1 Introduction ..... 148

6.7.2 Overview ..... 149

6.7.3 Features ..... 149

6.7.4 Memory map ..... 150

6.7.5 Register description ..... 150

6.7.6 Functional description ..... 153

6.7.7 Recommendations ..... 156

6.8 Clock monitor unit (CMU) ..... 156

6.8.1 Introduction ..... 156

6.8.2 Main features ..... 156

6.8.3 Block diagram ..... 157

6.8.4 Functional description ..... 158

6.8.5 Memory map and register description ..... 159

<b>7</b>	<b>Clock Generation Module (MC_CGM)</b>	<b>164</b>
7.1	Overview	164
7.2	Features	166
7.3	Modes of operation	166
7.3.1	Normal and reset modes of operation	166
7.4	External signal description	166
7.5	Memory map and register definition	166
7.5.1	Register descriptions	170
7.5.2	Output Clock Division Select Register (CGM_OCDS_SC)	171
7.5.3	System Clock Select Status Register (CGM_SC_SS)	172
7.6	Functional Description	174
7.6.1	System Clock Generation	174
7.6.2	Output Clock Multiplexing	175
7.6.3	Output Clock Division Selection	176
<b>8</b>	<b>Mode Entry Module (MC_ME)</b>	<b>177</b>
8.1	Overview	177
8.1.1	Features	179
8.1.2	Modes of Operation	179
8.2	External Signal Description	180
8.3	Memory Map and Register Definition	180
8.3.1	Register Description	188
8.4	Functional Description	209
8.4.1	Mode Transition Request	209
8.4.2	Modes Details	211
8.4.3	Mode transition process	215
8.4.4	Protection of Mode Configuration Registers	223
8.4.5	Mode Transition Interrupts	223
8.4.6	Peripheral Clock Gating	225
8.4.7	Application Example	226
<b>9</b>	<b>Reset Generation Module (MC_RGM)</b>	<b>228</b>
9.1	Introduction	228
9.1.1	Overview	228
9.1.2	Features	230
9.1.3	Modes of operation	230

- 9.2 External Signal Description ..... 231
- 9.3 Memory Map and Register Definition ..... 231
  - 9.3.1 Register descriptions ..... 234
- 9.4 Functional description ..... 244
  - 9.4.1 Reset State Machine ..... 244
  - 9.4.2 Destructive resets ..... 248
  - 9.4.3 External reset ..... 249
  - 9.4.4 Functional resets ..... 249
  - 9.4.5 STANDBY entry sequence ..... 249
  - 9.4.6 Alternate event generation ..... 250
  - 9.4.7 Boot mode capturing ..... 250
  
- 10 Power Control Unit (MC\_PCU) ..... 252**
  - 10.1 Introduction ..... 252
    - 10.1.1 Overview ..... 252
    - 10.1.2 Features ..... 253
    - 10.1.3 Modes of operation ..... 253
  - 10.2 External signal description ..... 254
  - 10.3 Memory map and register definition ..... 254
    - 10.3.1 Register descriptions ..... 255
  - 10.4 Functional description ..... 259
    - 10.4.1 General ..... 259
    - 10.4.2 Reset / Power-On Reset ..... 259
    - 10.4.3 MC\_PCU configuration ..... 259
    - 10.4.4 Mode transitions ..... 260
  - 10.5 Initialization information ..... 262
  - 10.6 Application information ..... 263
    - 10.6.1 STANDBY Mode Considerations ..... 263
  
- 11 Voltage Regulators and Power Supplies ..... 264**
  - 11.1 Voltage regulators ..... 264
    - 11.1.1 High power regulator (HPREG) ..... 264
    - 11.1.2 Low power regulator (LPREG) ..... 264
    - 11.1.3 Ultra low power regulator (ULPREG) ..... 264
    - 11.1.4 LVDs and POR ..... 265
    - 11.1.5 VREG digital interface ..... 265

11.1.6	Register description	265
11.2	Power supply strategy	266
11.3	Power domain organization	267
<b>12</b>	<b>Wakeup Unit (WKPU)</b>	<b>269</b>
12.1	Overview	269
12.2	Features	272
12.3	External signal description	273
12.4	Memory map and register description	273
12.4.1	Memory map	273
12.4.2	NMI Status Flag Register (NSR)	274
12.4.3	NMI Configuration Register (NCR)	274
12.4.4	Wakeup/Interrupt Status Flag Register (WISR)	276
12.4.5	Interrupt Request Enable Register (IRER)	276
12.4.6	Wakeup Request Enable Register (WRER)	277
12.4.7	Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)	277
12.4.8	Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)	277
12.4.9	Wakeup/Interrupt Filter Enable Register (WIFER)	278
12.4.10	Wakeup/Interrupt Pullup Enable Register (WIPUER)	278
12.5	Functional description	279
12.5.1	General	279
12.5.2	Non-maskable interrupts	279
12.5.3	External wakeups/interrupts	281
12.5.4	On-chip wakeups	282
<b>13</b>	<b>Real Time Clock / Autonomous Periodic Interrupt (RTC/API)</b>	<b>283</b>
13.1	Overview	283
13.2	Features	283
13.3	Device-specific information	285
13.4	Modes of operation	285
13.4.1	Functional mode	285
13.4.2	Debug mode	286
13.5	Register descriptions	286
13.5.1	RTC Supervisor Control Register (RTCSUPV)	286
13.5.2	RTC Control Register (RTCC)	287
13.5.3	RTC Status Register (RTCS)	289

	13.5.4	RTC Counter Register (RTCCNT)	290
	13.6	RTC functional description	290
<b>14</b>		<b>CAN Sampler</b>	<b>291</b>
	14.1	Introduction	291
	14.2	Main features	291
	14.3	Memory map and register description	292
	14.3.1	Control Register (CR)	292
	14.3.2	CAN Sampler Sample Registers 0–11	293
	14.4	Functional description	294
	14.4.1	Enabling/disabling the CAN sampler	294
	14.4.2	Selecting the Rx port	295
	14.4.3	Baudrate generation	295
<b>15</b>		<b>e200z0h Core</b>	<b>296</b>
	15.1	Overview	296
	15.2	Microarchitecture summary	296
	15.3	Block diagram	298
	15.4	Features	299
	15.4.1	Instruction unit features	299
	15.4.2	Integer unit features	300
	15.4.3	Load/Store unit features	300
	15.4.4	e200z0h system bus features	300
	15.4.5	Nexus 2+ features	300
	15.5	Core registers and programmer’s model	301
<b>16</b>		<b>Enhanced Direct Memory Access (eDMA)</b>	<b>303</b>
	16.1	Device-specific features	303
	16.2	Introduction	303
	16.2.1	Features	305
	16.3	Memory map and register definition	305
	16.3.1	Memory map	305
	16.3.2	Register descriptions	307
	16.4	Functional description	328
	16.4.1	eDMA basic data flow	330
	16.5	Initialization / application information	333



	16.5.1	eDMA initialization	333
	16.5.2	DMA programming errors	335
	16.5.3	DMA request assignments	336
	16.5.4	DMA arbitration mode considerations	337
	16.5.5	DMA transfer	337
	16.5.6	TCD status	340
	16.5.7	Channel linking	341
	16.5.8	Dynamic programming	342
<b>17</b>		<b>eDMA Channel Multiplexer (DMA_MUX)</b>	<b>345</b>
	17.1	Introduction	345
	17.2	Features	346
	17.3	Modes of operation	346
	17.4	External signal description	346
	17.5	Memory map and register definition	346
	17.5.1	Channel configuration registers (CHCONFIGn)	347
	17.6	DMA_MUX inputs	348
	17.6.1	DMA_MUX peripheral sources	348
	17.6.2	DMA_MUX periodic trigger inputs	350
	17.7	Functional description	350
	17.7.1	eDMA channels with periodic triggering capability	350
	17.7.2	eDMA channels with no triggering capability	352
	17.8	Initialization/Application information	353
	17.8.1	Reset	353
	17.8.2	Enabling and configuring sources	353
<b>18</b>		<b>Interrupt Controller (INTC)</b>	<b>356</b>
	18.1	Introduction	356
	18.2	Features	356
	18.3	Block diagram	358
	18.4	Modes of operation	358
	18.4.1	Normal mode	358
	18.5	Memory map and register description	360
	18.5.1	Module memory map	360
	18.5.2	Register description	360
	18.6	Functional description	368

18.6.1	Interrupt request sources	376
18.6.2	Priority management	377
18.6.3	Handshaking with processor	378
<b>18.7</b>	<b>Initialization/application information</b>	<b>380</b>
18.7.1	Initialization flow	380
18.7.2	Interrupt exception handler	381
18.7.3	ISR, RTOS, and task hierarchy	382
18.7.4	Order of execution	383
18.7.5	Priority ceiling protocol	384
18.7.6	Selecting priorities according to request rates and deadlines	385
18.7.7	Software configurable interrupt requests	385
18.7.8	Lowering priority within an ISR	386
18.7.9	Negating an interrupt request outside of its ISR	386
<b>19</b>	<b>Crossbar Switch (XBAR)</b>	<b>387</b>
19.1	Introduction	387
19.2	Block diagram	387
19.3	Overview	388
19.4	Features	388
19.5	Modes of operation	388
19.5.1	Normal mode	388
19.5.2	Debug mode	388
19.6	Functional description	388
19.6.1	Overview	388
19.6.2	General operation	389
19.6.3	Master ports	389
19.6.4	Slave ports	390
19.6.5	Priority assignment	390
19.6.6	Arbitration	390
<b>20</b>	<b>Memory Protection Unit (MPU)</b>	<b>392</b>
20.1	Introduction	392
20.2	Features	393
20.3	Modes of operation	394
20.4	External signal description	394
20.5	Memory map and register description	394

	20.5.1	Memory map	395
	20.5.2	Register description	396
	20.6	Functional description	407
	20.6.1	Access evaluation macro	407
	20.6.2	Putting it all together and AHB error terminations	408
	20.7	Initialization information	409
	20.8	Application information	409
<b>21</b>		<b>System Integration Unit Lite (SIUL)</b>	<b>411</b>
	21.1	Introduction	411
	21.2	Overview	411
	21.3	Features	413
	21.4	External signal description	413
	21.4.1	Detailed signal descriptions	414
	21.5	Memory map and register description	415
	21.5.1	SIUL memory map	415
	21.5.2	Register protection	416
	21.5.3	Register descriptions	416
	21.6	Functional description	435
	21.6.1	Pad control	435
	21.6.2	General purpose input and output pads (GPIO)	436
	21.6.3	External interrupts	437
	21.7	Pin muxing	438
<b>22</b>		<b>Inter-Integrated Circuit Bus Controller Module (I<sup>2</sup>C)</b>	<b>439</b>
	22.1	Introduction	439
	22.1.1	Overview	439
	22.1.2	Features	439
	22.1.3	Block diagram	439
	22.2	External signal description	440
	22.2.1	SCL	440
	22.2.2	SDA	440
	22.3	Memory map and register description	440
	22.3.1	Module memory map	440
	22.3.2	I <sup>2</sup> C Bus Address Register (IBAD)	441
	22.3.3	I <sup>2</sup> C Bus Frequency Divider Register (IBFD)	442

22.3.4	I <sup>2</sup> C Bus Control Register (IBCR)	448
22.3.5	I <sup>2</sup> C Bus Status Register (IBSR)	449
22.3.6	I <sup>2</sup> C Bus Data I/O Register (IBDR)	450
22.3.7	I <sup>2</sup> C Bus Interrupt Config Register (IBIC)	451
22.4	DMA Interface	451
22.5	Functional description	452
22.5.1	I-Bus protocol	452
22.5.2	Interrupts	456
22.6	Initialization/application information	457
22.6.1	I <sup>2</sup> C programming examples	457
<b>23</b>	<b>LIN Controller (LINFlex)</b>	<b>462</b>
23.1	Introduction	462
23.2	Main features	462
23.2.1	LIN mode features	462
23.2.2	UART mode features	462
23.2.3	Features common to LIN and UART	463
23.3	General description	463
23.4	Fractional baud rate generation	464
23.5	Operating modes	466
23.5.1	Initialization mode	466
23.5.2	Normal mode	466
23.5.3	Low power mode (Sleep)	467
23.6	Test modes	467
23.6.1	Loop Back mode	467
23.6.2	Self Test mode	467
23.7	Memory map and registers description	468
23.7.1	Memory map	468
23.8	Functional description	495
23.8.1	UART mode	495
23.8.2	LIN mode	497
23.8.3	8-bit timeout counter	506
23.8.4	Interrupts	508
<b>24</b>	<b>LIN Controller (LINFlexD)</b>	<b>509</b>
24.1	Introduction	509

24.2	Main features	509
24.2.1	LIN mode features	510
24.2.2	UART mode features	510
24.3	The LIN protocol	511
24.3.1	Dominant and recessive logic levels	511
24.3.2	LIN frames	511
24.3.3	LIN header	512
24.3.4	Response	513
24.4	LINFlexD and software intervention	514
24.5	Summary of operating modes	514
24.6	Controller-level operating modes	515
24.6.1	Initialization mode	515
24.6.2	Normal mode	516
24.6.3	Sleep (low-power) mode	516
24.7	LIN modes	516
24.7.1	Master mode	516
24.7.2	Slave mode	518
24.7.3	Slave mode with identifier filtering	521
24.7.4	Slave mode with automatic resynchronization	523
24.8	Test modes	525
24.8.1	Loop Back mode	525
24.8.2	Self Test mode	525
24.9	UART mode	526
24.9.1	Data frame structure	526
24.9.2	Buffer	527
24.9.3	UART transmitter	528
24.9.4	UART receiver	529
24.10	Memory map and register description	531
24.10.1	LIN control register 1 (LINC1)	533
24.10.2	LIN interrupt enable register (LINIER)	536
24.10.3	LIN status register (LINSR)	538
24.10.4	LIN error status register (LINESR)	541
24.10.5	UART mode control register (UARTCR)	542
24.10.6	UART mode status register (UARTSR)	545
24.10.7	LIN timeout control status register (LINTCSR)	547
24.10.8	LIN output compare register (LINOCCR)	548

- 24.10.9 LIN timeout control register (LINTOCR) ..... 549
- 24.10.10 LIN fractional baud rate register (LINFBR) ..... 550
- 24.10.11 LIN integer baud rate register (LINIBRR) ..... 550
- 24.10.12 LIN checksum field register (LINCFR) ..... 551
- 24.10.13 LIN control register 2 (LINC2) ..... 552
- 24.10.14 Buffer identifier register (BIDR) ..... 553
- 24.10.15 Buffer data register least significant (BDRL) ..... 554
- 24.10.16 Buffer data register most significant (BDRM) ..... 555
- 24.10.17 Identifier filter enable register (IFER) ..... 556
- 24.10.18 Identifier filter match index (IFMI) ..... 557
- 24.10.19 Identifier filter mode register (IFMR) ..... 558
- 24.10.20 Identifier filter control registers (IFCR0–IFCR15) ..... 559
- 24.10.21 Global control register (GCR) ..... 560
- 24.10.22 UART preset timeout register (UARTPTO) ..... 561
- 24.10.23 UART current timeout register (UARTCTO) ..... 562
- 24.10.24 DMA Tx enable register (DMATXE) ..... 563
- 24.10.25 DMA Rx enable register (DMARXE) ..... 563
- 24.11 DMA interface ..... 564
  - 24.11.1 Master node, TX mode ..... 564
  - 24.11.2 Master node, RX mode ..... 567
  - 24.11.3 Slave node, TX mode ..... 569
  - 24.11.4 Slave node, RX mode ..... 572
  - 24.11.5 UART node, TX mode ..... 575
  - 24.11.6 UART node, RX mode ..... 577
  - 24.11.7 Use cases and limitations ..... 580
- 24.12 Functional description ..... 581
  - 24.12.1 8-bit timeout counter ..... 581
  - 24.12.2 Interrupts ..... 582
  - 24.12.3 Fractional baud rate generation ..... 584
- 24.13 Programming considerations ..... 585
  - 24.13.1 Master node ..... 586
  - 24.13.2 Slave node ..... 587
  - 24.13.3 Extended frames ..... 590
  - 24.13.4 Timeout ..... 591
  - 24.13.5 UART mode ..... 591

**25 FlexCAN ..... 592**



25.1	Information specific to this device	592
25.1.1	Device-specific features	592
25.2	Introduction	592
25.2.1	Overview	593
25.2.2	FlexCAN module features	594
25.2.3	Modes of operation	595
25.3	External signal description	595
25.3.1	Overview	595
25.3.2	Signal descriptions	596
25.4	Memory map/register definition	596
25.4.1	FlexCAN memory mapping	596
25.4.2	Message Buffer Structure	598
25.4.3	Rx FIFO structure	601
25.4.4	Register descriptions	603
25.5	Functional description	623
25.5.1	Overview	623
25.5.2	Local Priority Transmission	624
25.5.3	Transmit process	624
25.5.4	Arbitration process	625
25.5.5	Receive process	625
25.5.6	Matching process	627
25.5.7	Data coherence	628
25.5.8	Rx FIFO	630
25.5.9	CAN Protocol Related Features	631
25.5.10	Modes of operation details	635
25.5.11	Interrupts	636
25.5.12	Bus interface	637
25.6	Initialization/application information	637
25.6.1	FlexCAN initialization sequence	637
25.6.2	FlexCAN Addressing and RAM size configurations	638
<b>26</b>	<b>Deserial Serial Peripheral Interface (DSPI)</b>	<b>639</b>
26.1	Introduction	639
26.2	Features	640
26.3	Modes of operation	641
26.3.1	Master mode	641

26.3.2	Slave mode	641
26.3.3	Module Disable mode	641
26.3.4	Debug mode	642
26.4	External signal description	642
26.4.1	Signal overview	642
26.4.2	Signal names and descriptions	642
26.5	Memory map and register description	643
26.5.1	Memory map	643
26.5.2	DSPI Module Configuration Register (DSPIx_MCR)	644
26.5.3	DSPI Transfer Count Register (DSPIx_TCR)	647
26.5.4	DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx_CTARn)	648
26.5.5	DSPI Status Register (DSPIx_SR)	656
26.5.6	DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)	658
26.5.7	DSPI PUSH TX FIFO Register (DSPIx_PUSHR)	660
26.5.8	DSPI POP RX FIFO Register (DSPIx_POPR)	662
26.5.9	DSPI Transmit FIFO Registers 0–3 (DSPIx_TXFRn)	663
26.6	Functional description	664
26.6.1	Modes of operation	665
26.6.2	Start and stop of DSPI transfers	666
26.6.3	Serial peripheral interface (SPI) configuration	667
26.6.4	DSPI baud rate and clock delay generation	670
26.6.5	Transfer formats	673
26.6.6	Continuous serial communications clock	680
26.6.7	Interrupt/DMA requests	682
26.6.8	Power saving features	684
26.7	Initialization and application information	684
26.7.1	How to change queues	684
26.7.2	Baud rate settings	685
26.7.3	Delay settings	686
26.7.4	Calculation of FIFO pointer addresses	686
<b>27</b>	<b>Timers</b>	<b>689</b>
27.1	Introduction	689
27.2	Technical overview	689
27.2.1	Overview of the STM	691
27.2.2	Overview of the eMIOS	691



27.2.3	Overview of the PIT .....	693
<b>27.3</b>	<b>System Timer Module (STM) .....</b>	<b>693</b>
27.3.1	Introduction .....	693
27.3.2	External signal description .....	694
27.3.3	Memory map and register definition .....	694
27.3.4	Functional description .....	697
<b>27.4</b>	<b>Enhanced Modular IO Subsystem (eMIOS) .....</b>	<b>698</b>
27.4.1	Introduction .....	698
27.4.2	External signal description .....	701
27.4.3	Memory map and register description .....	701
27.4.4	Functional description .....	713
27.4.5	Initialization/Application information .....	744
<b>27.5</b>	<b>Periodic Interrupt Timer (PIT) .....</b>	<b>748</b>
27.5.1	Introduction .....	748
27.5.2	Features .....	749
27.5.3	Signal description .....	749
27.5.4	Memory map and register description .....	749
27.5.5	Functional description .....	753
27.5.6	Initialization and application information .....	755
<b>28</b>	<b>Analog-to-Digital Converter (ADC) .....</b>	<b>756</b>
<b>28.1</b>	<b>Overview .....</b>	<b>756</b>
28.1.1	Device-specific features .....	756
28.1.2	Device-specific implementation .....	757
<b>28.2</b>	<b>Introduction .....</b>	<b>758</b>
<b>28.3</b>	<b>Functional description .....</b>	<b>758</b>
28.3.1	Analog channel conversion .....	758
28.3.2	Analog clock generator and conversion timings .....	761
28.3.3	ADC sampling and conversion timing .....	761
28.3.4	ADC CTU (Cross Triggering Unit) .....	765
28.3.5	Presampling .....	766
28.3.6	Programmable analog watchdog .....	767
28.3.7	DMA functionality .....	769
28.3.8	Interrupts .....	769
28.3.9	External decode signals delay .....	769
28.3.10	Power-down mode .....	770

28.3.11	Auto-clock-off mode	770
<b>28.4</b>	<b>Register descriptions</b>	<b>770</b>
28.4.1	Introduction	770
28.4.2	Control logic registers	777
28.4.3	Interrupt registers	781
28.4.4	DMA registers	788
28.4.5	Threshold registers	791
28.4.6	Presampling registers	793
28.4.7	Conversion timing registers CTR[0...2]	795
28.4.8	Mask registers	796
28.4.9	Delay registers	801
28.4.10	Data registers	802
28.4.11	Watchdog register	804
<b>29</b>	<b>Cross Triggering Unit (CTU)</b>	<b>816</b>
29.1	Introduction	816
29.2	Main features	816
29.3	Block diagram	816
29.4	Memory map and register descriptions	817
29.4.1	Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63)	817
29.5	Functional description	818
29.5.1	Channel value	820
<b>30</b>	<b>Flash Memory</b>	<b>823</b>
30.1	Introduction	823
30.2	Main features	824
30.3	Block diagram	824
30.4	Functional description	825
30.4.1	Module structure	825
30.4.2	Flash memory module sectorization	826
30.4.3	TestFlash block	827
30.4.4	Shadow sector	828
30.4.5	User mode operation	829
30.4.6	Reset	830
30.4.7	Power-down mode	830
30.4.8	Low power mode	831

30.5	Register description	831
30.5.1	CFlash register description	833
30.5.2	DFlash register description	869
30.6	Programming considerations	893
30.6.1	Modify operation	893
30.6.2	Double word program	894
30.6.3	Sector erase	896
30.7	Platform flash memory controller	904
30.7.1	Introduction	904
30.7.2	Memory map and register description	907
30.8	Functional description	916
30.8.1	Access protections	917
30.8.2	Read cycles – Buffer miss	917
30.8.3	Read cycles – Buffer hit	917
30.8.4	Write cycles	917
30.8.5	Error termination	918
30.8.6	Access pipelining	918
30.8.7	Flash error response operation	918
30.8.8	Bank0 page read buffers and prefetch operation	918
30.8.9	Bank1 Temporary Holding Register	920
30.8.10	Read-while-write functionality	921
30.8.11	Wait-state emulation	922
<b>31</b>	<b>Static RAM (SRAM)</b>	<b>924</b>
31.1	Introduction	924
31.2	Low power configuration	924
31.3	Register memory map	924
31.4	SRAM ECC mechanism	925
31.4.1	Access timing	925
31.4.2	Reset effects on SRAM accesses	926
31.5	Functional description	926
31.6	Initialization and application information	926
<b>32</b>	<b>Register Protection</b>	<b>928</b>
32.1	Introduction	928
32.2	Features	928

32.3	Modes of operation . . . . .	929
32.4	External signal description . . . . .	929
32.5	Memory map and register description . . . . .	929
32.5.1	Memory map . . . . .	930
32.5.2	Register description . . . . .	931
32.6	Functional description . . . . .	933
32.6.1	General . . . . .	933
32.6.2	Change lock settings . . . . .	933
32.6.3	Access errors . . . . .	936
32.7	Reset . . . . .	937
32.8	Protected registers . . . . .	937
<b>33</b>	<b>Software Watchdog Timer (SWT) . . . . .</b>	<b>944</b>
33.1	Overview . . . . .	944
33.2	Features . . . . .	944
33.3	Modes of operation . . . . .	944
33.4	External signal description . . . . .	944
33.5	Memory map and register description . . . . .	945
33.5.1	Memory map . . . . .	945
33.5.2	Register description . . . . .	945
33.6	Functional description . . . . .	950
<b>34</b>	<b>Error Correction Status Module (ECSM) . . . . .</b>	<b>952</b>
34.1	Introduction . . . . .	952
34.2	Overview . . . . .	952
34.3	Features . . . . .	952
34.4	Memory map and register description . . . . .	952
34.4.1	Memory map . . . . .	952
34.4.2	Register description . . . . .	953
34.4.3	Register protection . . . . .	973
<b>35</b>	<b>IEEE 1149.1 Test Access Port Controller (JTAGC) . . . . .</b>	<b>974</b>
35.1	Introduction . . . . .	974
35.2	Block diagram . . . . .	974
35.3	Overview . . . . .	974

35.4	Features	975
35.5	Modes of operation	975
35.5.1	Reset	975
35.5.2	IEEE 1149.1-2001 defined test modes	975
35.6	External signal description	976
35.7	Memory map and register description	976
35.7.1	Instruction Register	976
35.7.2	Bypass Register	977
35.7.3	Device Identification Register	977
35.7.4	Boundary Scan Register	978
35.8	Functional Description	978
35.8.1	JTAGC Reset Configuration	978
35.8.2	IEEE 1149.1-2001 (JTAG) Test Access Port	978
35.8.3	TAP controller state machine	978
35.8.4	JTAGC instructions	980
35.8.5	Boundary Scan	982
35.9	e200z0 OnCE controller	982
35.9.1	e200z0 OnCE Controller Block Diagram	982
35.9.2	e200z0 OnCE Controller Functional Description	983
35.9.3	e200z0 OnCE Controller Register Description	983
35.10	Initialization/application information	985
<b>36</b>	<b>Nexus Development Interface (NDI)</b>	<b>986</b>
36.1	Introduction	986
36.2	Block diagram	986
36.2.1	NDI features	989
36.2.2	Modes of operation	989
36.3	External signal description	990
36.4	Memory map and registers	991
36.5	NDI functional description	992
36.5.1	NPC_HNDSHK module	992
36.5.2	Enabling Nexus Clients for TAP Access	993
36.5.3	TAP Sharing	993
36.5.4	Configuring the NDI for Nexus Messaging	993
36.5.5	Programmable MCKO Frequency	994
36.6	Nexus Port Controller (NPC)	994

---

36.6.1	NPC Features	995
36.6.2	NPC Memory Map	995
36.6.3	NPC Register Descriptions	995
36.6.4	NPC Functional Description	998
36.6.5	NPC Initialization/Application Information	1003
36.7	Nexus Module (Nexus2+)	1003
36.7.1	Nexus2+ Introduction	1004
36.7.2	Nexus2+ Block Diagram	1004
36.7.3	Nexus2+ Features	1004
36.7.4	Enabling Nexus2+ Operation	1005
36.7.5	TCODEs Supported by Nexus2+	1005
36.7.6	Nexus2+ Memory Map	1009
36.7.7	Nexus2+ Register Definition	1009
36.7.8	Nexus2+ Register Access via JTAG / OnCE	1015
36.8	Debug Implementation	1016
36.9	Debug Capabilities	1016
<b>37</b>	<b>Revision History</b>	<b>1018</b>

## List of tables

Table 1.	Guide to this reference manual . . . . .	47
Table 2.	Reference manual integration and functional content . . . . .	54
Table 3.	SPC560B54/6x family comparison . . . . .	58
Table 4.	Supported eMIOS channel modes . . . . .	64
Table 5.	eMIOS configuration . . . . .	65
Table 6.	LINFlex numbering and naming . . . . .	74
Table 7.	SPC560B54/6x memory map . . . . .	76
Table 8.	Voltage supply pin descriptions . . . . .	83
Table 9.	System pin descriptions . . . . .	85
Table 10.	Functional port pin descriptions . . . . .	85
Table 11.	Nexus 2+ pin descriptions. . . . .	110
Table 12.	Boot mode selection . . . . .	111
Table 13.	RCHW field descriptions. . . . .	113
Table 14.	Examples of legal and illegal passwords . . . . .	115
Table 15.	Censorship configuration and truth table . . . . .	116
Table 16.	SSCM_STATUS[BMODE] values as used by BAM . . . . .	121
Table 17.	Serial boot mode – baud rates . . . . .	121
Table 18.	BAM censorship mode detection . . . . .	122
Table 19.	UART boot mode download protocol . . . . .	127
Table 20.	FlexCAN boot mode download protocol . . . . .	129
Table 21.	SSCM memory map . . . . .	131
Table 22.	SSCM_STATUS allowed register accesses . . . . .	132
Table 23.	SSCM_STATUS field descriptions . . . . .	132
Table 24.	SSCM_MEMCONFIG field descriptions . . . . .	133
Table 25.	SSCM_MEMCONFIG allowed register accesses. . . . .	133
Table 26.	SSCM_ERROR field descriptions. . . . .	134
Table 27.	SSCM_ERROR allowed register accesses . . . . .	134
Table 28.	SSCM_DEBUGPORT field descriptions. . . . .	135
Table 29.	Debug status port modes . . . . .	135
Table 30.	SSCM_DEBUGPORT allowed register accesses. . . . .	135
Table 31.	Password Comparison Register field descriptions . . . . .	136
Table 32.	SSCM_PWCMPH/L allowed register accesses . . . . .	137
Table 33.	SPC560B54/6x — Peripheral clock sources . . . . .	140
Table 34.	Truth table of crystal oscillator . . . . .	141
Table 35.	FXOSC_CTL field descriptions. . . . .	142
Table 36.	SXOSC truth table . . . . .	143
Table 37.	SXOSC_CTL field descriptions. . . . .	144
Table 38.	SIRC_CTL field descriptions. . . . .	146
Table 39.	FIRC_CTL field descriptions. . . . .	148
Table 40.	FMPLL memory map . . . . .	150
Table 41.	CR field descriptions. . . . .	150
Table 42.	Input divide ratios . . . . .	151
Table 43.	Output divide ratios. . . . .	152
Table 44.	Loop divide ratios . . . . .	152
Table 45.	MR field descriptions . . . . .	153
Table 46.	FMPLL lookup table . . . . .	154
Table 47.	Progressive clock switching on pll_select rising edge . . . . .	154
Table 48.	CMU memory map . . . . .	159

Table 49.	CMU_CSR field descriptions . . . . .	160
Table 50.	CMU_FDR field descriptions . . . . .	161
Table 51.	CMU_HFREFR field descriptions . . . . .	161
Table 52.	CMU_LFREFR field descriptions . . . . .	162
Table 53.	CMU_ISR field descriptions . . . . .	163
Table 54.	CMU_MDR field descriptions . . . . .	163
Table 55.	MC_CGM register description . . . . .	166
Table 56.	MC_CGM memory map . . . . .	167
Table 57.	Output Clock Enable Register (CGM_OC_EN) field descriptions. . . . .	171
Table 58.	Output Clock Division Select Register (CGM_OCDS_SC) field descriptions . . . . .	172
Table 59.	System Clock Select Status Register (CGM_SC_SS) field descriptions . . . . .	173
Table 60.	System Clock Divider Configuration Registers (CGM_SC_DC0...2) Field Descriptions . . . . .	173
Table 61.	MC_ME Mode Descriptions . . . . .	179
Table 62.	MC_ME Register Description . . . . .	181
Table 63.	MC_ME Memory Map . . . . .	183
Table 64.	Global Status Register (ME_GS) Field Descriptions . . . . .	189
Table 65.	Mode Control Register (ME_MCTL) Field Descriptions . . . . .	192
Table 66.	Mode Enable Register (ME_ME) Field Descriptions. . . . .	193
Table 67.	Interrupt Status Register (ME_IS) Field Descriptions . . . . .	194
Table 68.	Interrupt Mask Register (ME_IM) Field Descriptions . . . . .	195
Table 69.	Invalid Mode Transition Status Register (ME_IMTS) Field Descriptions . . . . .	196
Table 70.	Debug Mode Transition Status Register (ME_DMTS) Field Descriptions . . . . .	197
Table 71.	Mode Confutation Registers (ME_<mode>_MC) Field Descriptions . . . . .	203
Table 72.	Peripheral Status Registers 0...4 (ME_PS0...4) Field Descriptions. . . . .	207
Table 73.	Run Peripheral Configuration Registers (ME_RUN_PC0...7) Field Descriptions . . . . .	207
Table 74.	Low-Power Peripheral Configuration Registers (ME_LP_PC0...7) Field Descriptions . . . . .	208
Table 75.	Peripheral Control Registers (ME_PCTL0...143) Field Descriptions . . . . .	209
Table 76.	MC_ME Resource Control Overview . . . . .	216
Table 77.	MC_ME System Clock Selection Overview . . . . .	221
Table 78.	MC_RGM Register Description . . . . .	231
Table 79.	MC_RGM Memory Map . . . . .	232
Table 80.	Functional Event Status Register (RGM_FES) field descriptions. . . . .	234
Table 81.	Destructive Event Status Register (RGM_DES) field descriptions. . . . .	236
Table 82.	Functional Event Reset Disable Register (RGM_FERD) field descriptions . . . . .	237
Table 83.	Destructive Event Reset Disable Register (RGM_DERD) field descriptions . . . . .	238
Table 84.	Functional Event Alternate Request Register (RGM_FEAR) field descriptions . . . . .	239
Table 85.	Destructive Event Alternate Request Register (RGM_DEAR) field descriptions . . . . .	241
Table 86.	Functional Event Short Sequence Register (RGM_FESS) field descriptions. . . . .	242
Table 87.	STANDBY Reset Sequence Register (RGM_STDBY) field descriptions . . . . .	243
Table 88.	Functional Bidirectional Reset Enable Register (RGM_FBRE) field descriptions. . . . .	244
Table 89.	MC_RGM Reset Implications . . . . .	245
Table 90.	MC_RGM alternate event selection . . . . .	250
Table 91.	MC_PCU register description . . . . .	254
Table 92.	MC_PCU memory map. . . . .	254
Table 93.	Power Domain Configuration Register field descriptions . . . . .	256
Table 94.	Power Domain Status Register (PCU_PSTAT) field descriptions . . . . .	259
Table 95.	VREG_CTL field descriptions . . . . .	266
Table 96.	Wakeup vector mapping. . . . .	269
Table 97.	WKPU memory map. . . . .	273



Table 98.	NSR field descriptions	274
Table 99.	NCR field descriptions	275
Table 100.	WISR field descriptions	276
Table 101.	IRER field descriptions	276
Table 102.	WRER field descriptions	277
Table 103.	WIREER field descriptions	277
Table 104.	WIFEER field descriptions	278
Table 105.	WIFER field descriptions	278
Table 106.	WIPUER field descriptions	279
Table 107.	RTC/API register map	286
Table 108.	RTCSUPV field descriptions	286
Table 109.	RTCC field descriptions	287
Table 110.	RTCS field descriptions	289
Table 111.	RTCCNTfield descriptions	290
Table 112.	CAN Sampler memory map	292
Table 113.	CR field descriptions	292
Table 114.	Internal multiplexer correspondence	295
Table 115.	eDMA memory map	305
Table 116.	EDMA_CR field descriptions	308
Table 117.	EDMA_ESR field descriptions	310
Table 118.	EDMA_ERQRL field descriptions	312
Table 119.	EDMA_EEIRL field descriptions	313
Table 120.	EDMA_SERQR field descriptions	313
Table 121.	EDMA_CERQR field descriptions	314
Table 122.	EDMA_SEEIR field descriptions	314
Table 123.	EDMA_CEEIR field descriptions	315
Table 124.	EDMA_CIRQR field descriptions	315
Table 125.	EDMA_CER field descriptions	316
Table 126.	EDMA_SBR field descriptions	316
Table 127.	EDMA_CDSBR field descriptions	317
Table 128.	EDMA_IRQRL field descriptions	318
Table 129.	EDMA_ERL field descriptions	319
Table 130.	EDMA_HRSL field descriptions	320
Table 131.	EDMA_CPRn field descriptions	321
Table 132.	TCDn 32-bit memory structure	321
Table 133.	TCDn field descriptions	323
Table 134.	TCD primary control and status fields	334
Table 135.	DMA Request Summary for eDMA	336
Table 136.	Modulo Feature Example	340
Table 137.	Channel linking parameters	342
Table 138.	Coherency model for a dynamic channel link request	343
Table 139.	Coherency model for method 1	344
Table 140.	Coherency model for method 2	344
Table 141.	DMA_MUX memory map	347
Table 142.	CHCONFIGn field descriptions	347
Table 143.	Channel and trigger enabling	348
Table 144.	eDMA channel mapping	348
Table 145.	DMA_MUX periodic trigger inputs	350
Table 146.	Interrupt sources available	357
Table 147.	INTC memory map	360
Table 148.	INTC_MCR field descriptions	361
Table 149.	INTC_CPR field descriptions	362

Table 150.	PRI values	362
Table 151.	INTC_IACKR field descriptions	364
Table 152.	INTC_SSCIR[0:7] field descriptions	365
Table 153.	INTC_PSR0_3–INTC_PSR232_233 field descriptions	367
Table 154.	INTC Priority Select Register address offsets	367
Table 155.	Interrupt vector table	368
Table 156.	Order of ISR execution example	383
Table 157.	XBAR switch ports for SPC560B54/6x	387
Table 158.	Hardwired bus master priorities	390
Table 159.	MPU memory map	395
Table 160.	MPU_CESR field descriptions	396
Table 161.	SPERR implementation	397
Table 162.	MPU_EARn field descriptions	397
Table 163.	MPU_EDRn field descriptions	398
Table 164.	MPU_RGDn.Word0 field descriptions	400
Table 165.	MPU_RGDn.Word1 field descriptions	400
Table 166.	MPU_RGDn.Word2 field descriptions	401
Table 167.	MPU_RGDn.Word3 field descriptions	404
Table 168.	MPU_RGDAACn field descriptions	405
Table 169.	Protection violation definition	408
Table 170.	SIUL signal properties	413
Table 171.	SIUL memory map	415
Table 172.	MIDR1 field descriptions	417
Table 173.	MIDR2 field descriptions	418
Table 174.	ISR field descriptions	419
Table 175.	IRER field descriptions	420
Table 176.	IREER field descriptions	421
Table 177.	IFEER field descriptions	421
Table 178.	IFER field descriptions	422
Table 179.	PCR bit implementation by pad type	423
Table 180.	PCRx field descriptions	424
Table 181.	PSMI0_3 field descriptions	425
Table 182.	Peripheral input pin selection	426
Table 183.	GPDO0_3 field descriptions	430
Table 184.	GPDI0_3 field descriptions	431
Table 185.	PGPDO0 – PGPDO4 register map	432
Table 186.	PGPDI0 – PGPDI4 register map	432
Table 187.	MPGPDO0 – MPGPDO9 register map	433
Table 188.	MPGPDO0..MPGPDO9 field descriptions	434
Table 189.	IFMC field descriptions	435
Table 190.	IFCPR field descriptions	435
Table 191.	I2C memory map	441
Table 192.	IBAD field descriptions	441
Table 193.	IBFD field descriptions	442
Table 194.	I-Bus multiplier factor	442
Table 195.	I-Bus prescaler divider values	442
Table 196.	I-Bus tap and prescale values	443
Table 197.	I <sup>2</sup> C divider and hold values	445
Table 198.	IBCR field descriptions	448
Table 199.	IBSR Field Descriptions	449
Table 200.	IBIC field descriptions	451
Table 201.	Interrupt summary	456

Table 202.	Error calculation for programmed baud rates . . . . .	465
Table 203.	LINFlex memory map . . . . .	468
Table 204.	LINCR1 field descriptions . . . . .	470
Table 205.	Checksum bits configuration . . . . .	471
Table 206.	LIN master break length selection . . . . .	471
Table 207.	Operating mode selection . . . . .	472
Table 208.	LINIER field descriptions . . . . .	472
Table 209.	LINSR field descriptions . . . . .	475
Table 210.	LINESR field descriptions . . . . .	477
Table 211.	UARTCR field descriptions . . . . .	478
Table 212.	UARTSR field descriptions . . . . .	480
Table 213.	LINTCSR field descriptions . . . . .	482
Table 214.	LINOCR field descriptions . . . . .	483
Table 215.	LINTOCR field descriptions . . . . .	484
Table 216.	LINFBR field descriptions . . . . .	484
Table 217.	LINIBRR field descriptions . . . . .	485
Table 218.	Integer baud rate selection . . . . .	485
Table 219.	LINCFR field descriptions . . . . .	486
Table 220.	LINCR2 field descriptions . . . . .	487
Table 221.	BIDR field descriptions . . . . .	488
Table 222.	BDRL field descriptions . . . . .	489
Table 223.	BDRM field descriptions . . . . .	490
Table 224.	IFER field descriptions . . . . .	491
Table 225.	IFER[FACT] configuration . . . . .	491
Table 226.	IFMI field descriptions . . . . .	492
Table 227.	IFMR field descriptions . . . . .	493
Table 228.	IFMR[IFM] configuration . . . . .	493
Table 229.	IFCR $2n$ field descriptions . . . . .	494
Table 230.	IFCR $2n + 1$ field descriptions . . . . .	495
Table 231.	Message buffer . . . . .	496
Table 232.	Filter to interrupt vector correlation . . . . .	504
Table 233.	LINFlex interrupt control . . . . .	508
Table 234.	Errors in Master mode . . . . .	517
Table 235.	Errors in Slave mode . . . . .	520
Table 236.	Filter submodes . . . . .	521
Table 237.	Filter to interrupt vector correlation . . . . .	523
Table 238.	UART buffer structure . . . . .	528
Table 239.	BDRL access in UART mode . . . . .	528
Table 240.	BDRM access in UART mode . . . . .	529
Table 241.	UART receiver scenarios . . . . .	530
Table 242.	LINFlexD_0 memory map . . . . .	531
Table 243.	LINFlexD_1 memory map . . . . .	532
Table 244.	LINCR1 field descriptions . . . . .	533
Table 246.	LIN master break length selection . . . . .	535
Table 247.	Operating mode selection . . . . .	535
Table 245.	Checksum bits configuration . . . . .	535
Table 248.	LINIER field descriptions . . . . .	536
Table 249.	LINSR field descriptions . . . . .	539
Table 250.	LINESR field descriptions . . . . .	541
Table 251.	UARTCR field descriptions . . . . .	543
Table 252.	UARTSR field descriptions . . . . .	545
Table 253.	LINTCSR field descriptions . . . . .	547

Table 254.	LINOCR field descriptions	548
Table 255.	LINTOCR field descriptions	549
Table 256.	LINFBR field descriptions	550
Table 257.	LINIBRR field descriptions	551
Table 258.	Integer baud rate selection	551
Table 259.	LINCFR field descriptions	551
Table 260.	LINCR2 field descriptions	552
Table 261.	BIDR field descriptions	554
Table 262.	BDRL field descriptions	554
Table 263.	BDRM field descriptions	555
Table 264.	IFER field descriptions	556
Table 265.	IFER[FACT] configuration	556
Table 266.	IFMI field descriptions	557
Table 267.	IFMR field descriptions	558
Table 268.	IFMR[IFM] configuration	558
Table 269.	IFCR functionality based on mode	559
Table 270.	IFCR field descriptions	559
Table 271.	GCR field descriptions	560
Table 272.	UARTPTO field descriptions	562
Table 273.	UARTCTO field descriptions	563
Table 274.	DMATXE field descriptions	563
Table 275.	DMARXE field descriptions	564
Table 276.	Register settings (master node, TX mode)	565
Table 277.	TCD settings (master node, TX mode)	567
Table 278.	TCD settings (master node, RX mode)	569
Table 279.	Register settings (slave node, TX mode)	570
Table 280.	TCD settings (slave node, TX mode)	572
Table 281.	Register settings (slave node, RX mode)	573
Table 282.	TCD settings (slave node, RX mode)	574
Table 283.	TCD settings (UART node, TX mode)	577
Table 284.	TCD settings (UART node, RX mode)	580
Table 285.	LINFlexD interrupt control	582
Table 286.	Error calculation for programmed baud rates	585
Table 287.	FlexCAN Signals	596
Table 288.	FlexCAN memory map	597
Table 289.	Message Buffer MB0 memory mapping	598
Table 290.	Message Buffer Structure field description	598
Table 291.	Message Buffer Code for Rx buffers	599
Table 292.	Message Buffer Code for Tx buffers	600
Table 293.	Rx FIFO Structure field description	603
Table 294.	MCR field descriptions	604
Table 295.	IDAM coding	607
Table 296.	CTRL field descriptions	608
Table 297.	TIMER field descriptions	611
Table 298.	RXGMASK field descriptions	613
Table 299.	ECR field descriptions	615
Table 300.	ESR field descriptions	616
Table 301.	Fault confinement state	618
Table 302.	MASK2 field descriptions	619
Table 303.	IMASK1 field descriptions	620
Table 304.	IFLAG2 field descriptions	621
Table 305.	IFLAG1 field descriptions	622

Table 306.	RXIMR0–RXIMR63 field descriptions . . . . .	623
Table 307.	Time Segment Syntax . . . . .	634
Table 308.	Bosch CAN 2.0B standard compliant bit time segment settings . . . . .	634
Table 309.	Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate . . . . .	635
Table 310.	Signal properties . . . . .	642
Table 311.	DSPI memory map . . . . .	644
Table 312.	DSPIx_MCR field descriptions . . . . .	645
Table 313.	DSPIx_TCR field descriptions . . . . .	648
Table 314.	DSPIx_CTARn field descriptions . . . . .	649
Table 315.	DSPI SCK duty cycle . . . . .	652
Table 316.	DSPI transfer frame size . . . . .	652
Table 317.	DSPI PCS to SCK delay scaler . . . . .	652
Table 318.	DSPI After SCK delay scaler . . . . .	653
Table 319.	DSPI delay after transfer scaler . . . . .	653
Table 320.	DSPI baud rate scaler . . . . .	653
Table 321.	DSPI SCK duty cycle . . . . .	654
Table 322.	DSPI transfer frame size . . . . .	654
Table 323.	DSPI PCS to SCK delay scaler . . . . .	654
Table 324.	DSPI After SCK delay scaler . . . . .	655
Table 325.	DSPI delay after transfer scaler . . . . .	655
Table 326.	DSPI baud rate scaler . . . . .	655
Table 327.	DSPIx_SR field descriptions . . . . .	656
Table 328.	DSPIx_RSER field descriptions . . . . .	659
Table 329.	DSPIx_PUSHR field descriptions . . . . .	661
Table 330.	DSPIx_POPR field descriptions . . . . .	662
Table 331.	DSPIx_TXFRn field descriptions . . . . .	663
Table 332.	DSPIx_RXFRn field description . . . . .	664
Table 333.	State transitions for start and stop of DSPI transfers . . . . .	667
Table 334.	Baud rate computation example . . . . .	671
Table 335.	CS to SCK delay computation example . . . . .	671
Table 336.	After SCK delay computation example . . . . .	671
Table 337.	Delay after transfer computation example . . . . .	672
Table 338.	Peripheral chip select strobe assert computation example . . . . .	673
Table 339.	Peripheral chip select strobe negate computation example . . . . .	673
Table 340.	Delayed master sample point . . . . .	676
Table 341.	Interrupt and DMA Request Conditions . . . . .	682
Table 342.	Baud rate values . . . . .	685
Table 343.	Delay values . . . . .	686
Table 344.	eMIOS_0 channel to pin mapping . . . . .	692
Table 345.	eMIOS_1 channel to pin mapping . . . . .	692
Table 346.	STM memory map . . . . .	694
Table 347.	STM_CR field descriptions . . . . .	695
Table 348.	STM_CNT field descriptions . . . . .	696
Table 349.	STM_CCRn field descriptions . . . . .	696
Table 350.	STM_CIRn field descriptions . . . . .	697
Table 351.	STM_CMPn field descriptions . . . . .	697
Table 352.	eMIOS memory map . . . . .	701
Table 353.	Unified Channel memory map . . . . .	702
Table 354.	EMIOSMCR field descriptions . . . . .	702
Table 355.	Global prescaler clock divider . . . . .	703
Table 356.	EMIOSGFLAG field descriptions . . . . .	704
Table 357.	EMIOSOUDIS field descriptions . . . . .	705

Table 358.	EMIOSUCDIS field descriptions . . . . .	705
Table 359.	EMIOSA[n], EMIOSB[n] and EMIOSALTA[n] values assignment. . . . .	707
Table 360.	EMIOSC[n] field descriptions . . . . .	708
Table 361.	UC internalprescaler clock divider . . . . .	710
Table 362.	UC input filter bits . . . . .	711
Table 363.	UC BSL bits . . . . .	711
Table 364.	Channel mode selection . . . . .	711
Table 365.	EMIOSS[n] field descriptions . . . . .	712
Table 366.	PIT memory map . . . . .	749
Table 367.	Timer channel <i>n</i> . . . . .	749
Table 368.	PITMCR field descriptions . . . . .	750
Table 369.	LDVAL field descriptions. . . . .	751
Table 370.	CVAL field descriptions. . . . .	752
Table 371.	TCTRL field descriptions . . . . .	752
Table 372.	TFLG field descriptions. . . . .	753
Table 373.	ADC sampling and conversion timing at 5 V / 3.3 V for ADC_0 . . . . .	763
Table 374.	Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V for ADC_0 . . . . .	763
Table 375.	ADC sampling and conversion timing at 5 V for ADC_1 . . . . .	764
Table 376.	ADC sampling and conversion timing at 3.3 V for ADC_1 . . . . .	764
Table 377.	Max/Min ADC_clk frequency and related configuration settings at 5 V for ADC_1 . . . . .	765
Table 378.	Max/Min ADC_clk frequency and related configuration settings at 3.3 V for ADC_1 . . . . .	765
Table 379.	Presampling voltage selection based on PREVALx fields . . . . .	767
Table 380.	Values of WDGxH and WDGxL fields. . . . .	768
Table 381.	10-bit ADC_0 digital registers . . . . .	771
Table 382.	12-bit ADC_1 digital registers . . . . .	775
Table 383.	MCR field descriptions . . . . .	778
Table 384.	MSR field descriptions . . . . .	780
Table 385.	ISR field descriptions . . . . .	781
Table 386.	CEOCFR[0...2] register description . . . . .	781
Table 387.	CEOCFR field descriptions. . . . .	783
Table 388.	Interrupt Mask Register (IMR) field descriptions. . . . .	784
Table 389.	CIMR[0...2] register description . . . . .	784
Table 390.	CIMR field descriptions. . . . .	786
Table 391.	ADC_0 WTISR field descriptions . . . . .	786
Table 392.	ADC_1 WTISR field descriptions . . . . .	787
Table 393.	ADC_0 WTIMR field descriptions . . . . .	788
Table 394.	ADC_1 WTIMR field descriptions . . . . .	788
Table 395.	DMAE field descriptions . . . . .	789
Table 396.	DMAR[0...2] register description . . . . .	789
Table 397.	DMARx field descriptions . . . . .	791
Table 398.	ADC_0 THRHLR field descriptions . . . . .	792
Table 399.	ADC_1 THRHLR field descriptions . . . . .	792
Table 400.	PSCR field descriptions . . . . .	793
Table 401.	PSR[0...2] register description . . . . .	793
Table 402.	PSR field descriptions. . . . .	795
Table 403.	CTR[0...2] register description . . . . .	795
Table 404.	CTR field descriptions. . . . .	796
Table 405.	NCMR[0..2] register description . . . . .	797
Table 406.	NCMR field descriptions . . . . .	798
Table 407.	JCMR[0...2] register description . . . . .	799
Table 408.	JCMR field descriptions . . . . .	800
Table 409.	DSDR field descriptions . . . . .	801

Table 410.	PDEDL field descriptions . . . . .	802
Table 411.	CDR[0...95] register description . . . . .	802
Table 412.	CDR field descriptions . . . . .	804
Table 413.	CWSELR[0...11] register description . . . . .	804
Table 414.	CWSELR field descriptions – ADC_0 . . . . .	805
Table 415.	CWSELR1 field descriptions – ADC_0 . . . . .	806
Table 416.	CWSELR4 field descriptions – ADC_0 . . . . .	806
Table 417.	CWSELR5 field descriptions – ADC_0 . . . . .	806
Table 418.	CWSELR6 field descriptions – ADC_0 . . . . .	807
Table 419.	CWSELR7 field descriptions – ADC_0 . . . . .	807
Table 420.	CWSELR8 field descriptions – ADC_0 . . . . .	808
Table 421.	CWSELR9 field descriptions – ADC_0 . . . . .	808
Table 422.	CWSELR10 field descriptions – ADC_0 . . . . .	809
Table 423.	CWSELR11 field descriptions – ADC_0 . . . . .	809
Table 424.	CWSELR0 field descriptions – ADC_1 . . . . .	810
Table 425.	CWSELR1 field descriptions – ADC_1 . . . . .	810
Table 426.	CWSELR4 field descriptions – ADC_1 . . . . .	811
Table 427.	CWSELR5 field descriptions – ADC_1 . . . . .	811
Table 428.	CWENR[0...2] register description . . . . .	812
Table 429.	CWENRx field descriptions . . . . .	813
Table 430.	AWORRx field descriptions . . . . .	815
Table 431.	CTU memory map . . . . .	817
Table 432.	CTU_EVTCFGRx field descriptions . . . . .	817
Table 433.	Trigger source . . . . .	818
Table 434.	CTU-to-ADC channel assignment . . . . .	820
Table 435.	Flash memory features . . . . .	824
Table 436.	CFlash module sectorization . . . . .	826
Table 437.	DFlash module sectorization . . . . .	827
Table 438.	CFlash TestFlash structure . . . . .	828
Table 439.	DFlash TestFlash structure . . . . .	828
Table 440.	Shadow sector structure . . . . .	829
Table 441.	CFlash registers . . . . .	831
Table 442.	DFlash registers . . . . .	832
Table 443.	CFLASH_MCR field descriptions . . . . .	834
Table 444.	Array space size . . . . .	837
Table 445.	Low address space configuration . . . . .	838
Table 446.	Mid address space configuration . . . . .	838
Table 447.	CFLASH_MCR bits set/clear priority levels . . . . .	838
Table 448.	CFLASH_LML field descriptions . . . . .	840
Table 449.	CFLASH_NVLML field descriptions . . . . .	842
Table 450.	CFLASH_HBL field descriptions . . . . .	844
Table 451.	CFLASH_NVHBL field descriptions . . . . .	846
Table 452.	CFLASH_SLL field descriptions . . . . .	848
Table 453.	CFLASH_NVSLI field descriptions . . . . .	851
Table 454.	CFLASH_LMS field descriptions . . . . .	854
Table 455.	CFLASH_HBS field descriptions . . . . .	855
Table 456.	CFLASH_ADR field descriptions . . . . .	856
Table 457.	CFLASH_ADR content: priority list . . . . .	856
Table 458.	CFLASH_UT0 field descriptions . . . . .	857
Table 459.	CFLASH_UT1 field descriptions . . . . .	859
Table 460.	CFLASH_UT2 field descriptions . . . . .	860
Table 461.	CFLASH_UMISR0 field descriptions . . . . .	861

Table 462.	CFLASH_UMISR1 field descriptions	862
Table 463.	CFLASH_UMISR2 field descriptions	863
Table 464.	CFLASH_UMISR3 field descriptions	864
Table 465.	CFLASH_UMISR4 field descriptions	865
Table 466.	NVPWD0 field descriptions	866
Table 467.	NVPWD1 field descriptions	867
Table 468.	NVSCC0 field descriptions	867
Table 469.	NVSCC1 field descriptions	868
Table 470.	NVUSRO field descriptions	869
Table 471.	DFLASH_MCR field descriptions	870
Table 472.	Array space size	873
Table 473.	Low address space configuration	874
Table 474.	Mid address space configuration	874
Table 475.	DFLASH_MCR bits set/clear priority levels	874
Table 476.	DFLASH_LML field descriptions	876
Table 477.	DFLASH_NVLML field descriptions	878
Table 478.	DFLASH_SLL field descriptions	880
Table 479.	DFLASH_NVSLM field descriptions	882
Table 480.	DFLASH_LMS field descriptions	883
Table 481.	DFLASH_ADR field descriptions	884
Table 482.	DFLASH_ADR content: priority list	884
Table 483.	DFLASH_UT0 field descriptions	885
Table 484.	DFLASH_UT1 field descriptions	887
Table 485.	DFLASH_UT2 field descriptions	888
Table 486.	DFLASH_UMISR0 field descriptions	889
Table 487.	DFLASH_UMISR1 field descriptions	890
Table 488.	DFLASH_UMISR2 field descriptions	891
Table 489.	DFLASH_UMISR3 field descriptions	892
Table 490.	DFLASH_UMISR4 field descriptions	893
Table 491.	Flash memory modify operations	894
Table 492.	Bit manipulation: Double words with the same ECC value	902
Table 493.	Flash memory-related regions in the system memory map	908
Table 494.	Platform flash memory controller 32-bit memory map	909
Table 495.	PFCR0 field descriptions	910
Table 496.	PFCR1 field descriptions	913
Table 497.	PFAPR field descriptions	915
Table 498.	NVPFAPR field descriptions	916
Table 499.	Platform flash memory controller stall-while-write interrupts	922
Table 500.	Additional wait-state encoding	923
Table 501.	Extended additional wait-state encoding	923
Table 502.	SRAM behavior in chip modes	924
Table 503.	Low power configuration	924
Table 504.	SRAM memory map	924
Table 505.	Number of wait states required for SRAM operations	926
Table 506.	Register protection memory map	930
Table 507.	SLBR $n$ field descriptions	931
Table 508.	Soft lock bits vs. protected address	932
Table 509.	GCR field descriptions	933
Table 510.	Protected registers	937
Table 511.	SWT memory map	945
Table 512.	SWT_CR field descriptions	946
Table 513.	SWT_IR field descriptions	948



Table 514.	SWT_TO Register field descriptions . . . . .	948
Table 515.	SWT_WN Register field descriptions . . . . .	949
Table 516.	SWT_SR field descriptions . . . . .	949
Table 517.	SWT_CO field descriptions . . . . .	950
Table 518.	ECSM memory map . . . . .	952
Table 519.	PCT field descriptions . . . . .	954
Table 520.	REV field descriptions . . . . .	954
Table 521.	IOPMC field descriptions . . . . .	955
Table 522.	MWCR field descriptions . . . . .	956
Table 523.	MIR field descriptions . . . . .	957
Table 524.	MUDCR field descriptions . . . . .	958
Table 525.	ECR field descriptions . . . . .	960
Table 526.	ESR field descriptions . . . . .	962
Table 527.	EEGR field descriptions . . . . .	963
Table 528.	PFEAR field descriptions . . . . .	966
Table 529.	PFEMR field descriptions . . . . .	966
Table 530.	PFEAT field descriptions . . . . .	967
Table 531.	PFEDR field descriptions . . . . .	968
Table 532.	PREAR field descriptions . . . . .	969
Table 533.	PRESR field descriptions . . . . .	969
Table 534.	RAM syndrome mapping for single-bit correctable errors . . . . .	969
Table 535.	PREMR field descriptions . . . . .	971
Table 536.	PREAT field descriptions . . . . .	972
Table 537.	PREDR field descriptions . . . . .	973
Table 538.	JTAG signal properties . . . . .	976
Table 539.	Device Identification Register Field Descriptions . . . . .	977
Table 540.	JTAG Instructions . . . . .	980
Table 541.	e200z0 OnCE Register Addressing . . . . .	984
Table 542.	Signal properties . . . . .	990
Table 543.	Nexus Client JTAG Instructions . . . . .	991
Table 544.	NDI Registers . . . . .	992
Table 545.	NDI Configuration Options . . . . .	994
Table 546.	MCKO_DIV Values . . . . .	994
Table 547.	NPC Memory Map . . . . .	995
Table 548.	DID Field Descriptions . . . . .	996
Table 549.	PCR Field Descriptions . . . . .	997
Table 550.	NPC Output Messages . . . . .	999
Table 551.	Loading NEXUS-ENABLE Instruction . . . . .	1001
Table 552.	Write to a 32-Bit Nexus Client Register . . . . .	1002
Table 553.	Public TCODEs Supported by Nexus2+ . . . . .	1005
Table 554.	Error Code Encoding (TCODE = 8) . . . . .	1008
Table 555.	RCODE values (TCODE = 27) . . . . .	1008
Table 556.	Event Code Encoding (TCODE = 33) . . . . .	1008
Table 557.	Nexus2+ Memory Map . . . . .	1009
Table 558.	DC1 Field Descriptions . . . . .	1010
Table 559.	DC2 Field Descriptions . . . . .	1011
Table 560.	DS Field Descriptions . . . . .	1011
Table 561.	RWCS Field Description . . . . .	1012
Table 562.	Read/Write Access Status Bit Encoding . . . . .	1013
Table 563.	RWD Data Placement for Transfers . . . . .	1014
Table 564.	RWD data placement for Transfers . . . . .	1014
Table 565.	WT Field Descriptions . . . . .	1015

---

Table 566. Nexus Debug Requirements . . . . .	1016
Table 567. Document revision history . . . . .	1018

## List of figures

Figure 1.	Register figure conventions	52
Figure 2.	SPC560B54/6x block diagram	60
Figure 3.	100-pin LQFP pinout	79
Figure 4.	144-pin LQFP pinout	80
Figure 5.	176-pin LQFP pinout	81
Figure 6.	208-pin BGA pinout	82
Figure 7.	Boot mode selection	112
Figure 8.	Boot sector structure	113
Figure 9.	Flash memory boot mode sequence	114
Figure 10.	Censorship control in flash memory boot mode	118
Figure 11.	Censorship control in serial boot mode	119
Figure 12.	BAM logic flow	120
Figure 13.	BAM censorship mode detection	123
Figure 14.	BAM serial boot mode flow for censorship enabled and private password	125
Figure 15.	Start address, VLE bit and download size in bytes	126
Figure 16.	LINFlex bit timing in UART mode	127
Figure 17.	FlexCAN bit timing	129
Figure 18.	SSCM block diagram	130
Figure 19.	System Status Register (SSCM_STATUS)	131
Figure 20.	System Memory Configuration Register (SSCM_MEMCONFIG)	132
Figure 21.	Error Configuration (SSCM_ERROR)	133
Figure 22.	Debug Status Port Register (SSCM_DEBUGPORT)	134
Figure 23.	Password Comparison Register High Word (SSCM_PWCMPH)	136
Figure 24.	Password Comparison Register Low Word (SSCM_PWC MPL)	136
Figure 25.	SPC560B54/6x system clock generation	139
Figure 26.	Fast External Crystal Oscillator Control Register (FXOSC_CTL)	142
Figure 27.	Slow External Crystal Oscillator Control Register (SXOSC_CTL)	144
Figure 28.	Low Power RC Control Register (SIRC_CTL)	146
Figure 29.	FIRC Oscillator Control Register (FIRC_CTL)	148
Figure 30.	FMPLL block diagram	149
Figure 31.	Control Register (CR)	150
Figure 32.	Modulation Register (MR)	152
Figure 33.	FMPLL output clock division flow during progressive switching	154
Figure 34.	Frequency modulation	155
Figure 35.	Clock Monitor Unit diagram	157
Figure 36.	Control Status Register (CMU_CSR)	160
Figure 37.	Frequency Display Register (CMU_FDR)	161
Figure 38.	High Frequency Reference Register FMPLL (CMU_HFREFR)	161
Figure 39.	Low Frequency Reference Register FMPLL (CMU_LFREFR)	162
Figure 40.	Interrupt status register (CMU_ISR)	162
Figure 41.	Measurement Duration Register (CMU_MDR)	163
Figure 42.	MC_CGM block diagram	165
Figure 43.	Output Clock Enable Register (CGM_OC_EN)	171
Figure 44.	Output Clock Division Select Register (CGM_OCDS_SC)	171
Figure 45.	System Clock Select Status Register (CGM_SC_SS)	172
Figure 46.	System Clock Divider Configuration Registers (CGM_SC_DC0...2)	173
Figure 47.	MC_CGM System Clock Generation Overview	175
Figure 48.	MC_CGM Output Clock Multiplexer and PA[0] Generation	176

Figure 49.	MC_ME block diagram	178
Figure 50.	Global Status Register (ME_GS)	189
Figure 51.	Mode Control Register (ME_MCTL)	191
Figure 52.	Mode Enable Register (ME_ME)	192
Figure 53.	Interrupt Status Register (ME_IS)	194
Figure 54.	Interrupt Mask Register (ME_IM)	195
Figure 55.	Invalid Mode Transition Status Register (ME_IMTS)	196
Figure 56.	Debug Mode Transition Status Register (ME_DMTS)	197
Figure 57.	Invalid Mode Transition Status Register (ME_IMTS)	199
Figure 58.	TEST Mode Configuration Register (ME_TEST_MC)	200
Figure 59.	SAFE Mode Configuration Register (ME_SAFE_MC)	200
Figure 60.	DRUN Mode Configuration Register (ME_DRUN_MC)	201
Figure 61.	RUN0...3 Mode Configuration Registers (ME_RUN0...3_MC)	201
Figure 62.	HALT Mode Configuration Register (ME_HALT_MC)	202
Figure 63.	STOP Mode Configuration Register (ME_STOP_MC)	202
Figure 64.	STANDBY Mode Configuration Register (ME_STANDBY_MC)	203
Figure 65.	Peripheral Status Register 0 (ME_PS0)	205
Figure 66.	Peripheral Status Register 1 (ME_PS1)	205
Figure 67.	Peripheral Status Register 2 (ME_PS2)	206
Figure 68.	Peripheral Status Register 3 (ME_PS3)	206
Figure 69.	Run Peripheral Configuration Registers (ME_RUN_PC0...7)	207
Figure 70.	Low-Power Peripheral Configuration Registers (ME_LP_PC0...7)	208
Figure 71.	Peripheral Control Registers (ME_PCTL0...143)	209
Figure 72.	MC_ME Mode Diagram	210
Figure 73.	MC_ME Application Example Flow Diagram	227
Figure 74.	MC_RGM Block Diagram	229
Figure 75.	Functional Event Status Register (RGM_FES)	234
Figure 76.	Destructive Event Status Register (RGM_DES)	235
Figure 77.	Functional Event Reset Disable Register (RGM_FERD)	237
Figure 78.	Destructive Event Reset Disable Register (RGM_DERD)	238
Figure 79.	Functional Event Alternate Request Register (RGM_FEAR)	239
Figure 80.	Destructive Event Alternate Request Register (RGM_DEAR)	240
Figure 81.	Functional Event Short Sequence Register (RGM_FESS)	241
Figure 82.	STANDBY Reset Sequence Register (RGM_STDBY)	243
Figure 83.	Functional Bidirectional Reset Enable Register (RGM_FBRE)	243
Figure 84.	MC_RGM State Machine	246
Figure 85.	Reset configuration timing	247
Figure 86.	MC_PCU Block Diagram	253
Figure 87.	Power Domain #0 Configuration Register (PCU_PCONF0)	256
Figure 88.	Power Domain #1 Configuration Register (PCU_PCONF1)	257
Figure 89.	Power Domain #2 Configuration Register (PCU_PCONF2)	258
Figure 90.	Power Domain #3 Configuration Register (PCU_PCONF3)	258
Figure 91.	Power Domain Status Register (PCU_PSTAT)	259
Figure 92.	MC_PCU Events During Power Sequences (non-STANDBY mode)	260
Figure 93.	MC_PCU Events During Power Sequences (STANDBY mode)	262
Figure 94.	Voltage Regulator Control Register (VREG_CTL)	266
Figure 95.	Power domain organization	268
Figure 96.	WKPU block diagram	272
Figure 97.	NMI Status Flag Register (NSR)	274
Figure 98.	NMI Configuration Register (NCR)	275
Figure 99.	Wakeup/Interrupt Status Flag Register (WISR)	276
Figure 100.	Interrupt Request Enable Register (IRER)	276

Figure 101. Wakeup Request Enable Register (WRER) . . . . .	277
Figure 102. Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER) . . . . .	277
Figure 103. Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER) . . . . .	278
Figure 104. Wakeup/Interrupt Filter Enable Register (WIFER) . . . . .	278
Figure 105. Wakeup/Interrupt Pullup Enable Register (WIPUER) . . . . .	279
Figure 106. NMI pad diagram . . . . .	280
Figure 107. External interrupt pad diagram . . . . .	281
Figure 108. RTC/API block diagram . . . . .	284
Figure 109. Clock gating for RTC clocks . . . . .	285
Figure 110. RTC Supervisor Control Register (RTCSUPV) . . . . .	286
Figure 111. RTC Control Register (RTCC) . . . . .	287
Figure 112. RTC Status Register (RTCS) . . . . .	289
Figure 113. RTC Counter Register (RTCCNT) . . . . .	290
Figure 114. Extended CAN data frame . . . . .	291
Figure 115. Control Register (CR) . . . . .	292
Figure 116. CAN Sampler Sample Registers 0–11 . . . . .	293
Figure 117. e200z0h block diagram . . . . .	298
Figure 118. e200z0 SUPERVISOR Mode Program Model SPRs . . . . .	302
Figure 119. eDMA block diagram . . . . .	304
Figure 120. DMA Control Register (EDMA_CR) . . . . .	308
Figure 121. DMA Error Status (EDMA_ESR) Register . . . . .	310
Figure 122. DMA Enable Request (EDMA_ERQRL) Registers . . . . .	312
Figure 123. DMA Enable Error Interrupt (EDMA_EEIRL) Register . . . . .	313
Figure 124. DMA Set Enable Request (EDMA_SERQR) Register . . . . .	313
Figure 125. DMA Clear Enable Request (EDMA_CERQR) Register . . . . .	314
Figure 126. DMA Set Enable Error Interrupt (EDMA_SEEIR) Register . . . . .	314
Figure 127. DMA Clear Enable Error Interrupt (EDMA_CEEIR) Register . . . . .	315
Figure 128. DMA Clear Interrupt Request (EDMA_CIRQR) Fields . . . . .	315
Figure 129. DMA Clear Error (EDMA_CER) Register . . . . .	316
Figure 130. DMA Set START Bit (EDMA_SBR) Register . . . . .	316
Figure 131. DMA Clear DONE Status (EDMA_CDSBR) Register . . . . .	317
Figure 132. DMA Interrupt Request (EDMA_IRQRL) Registers . . . . .	318
Figure 133. DMA Error (EDMA_ERL) Registers . . . . .	319
Figure 134. DMA Hardware Request Status (EDMA_HRSL) Register . . . . .	319
Figure 135. DMA Channel n Priority (EDMA_CPRn) Register . . . . .	320
Figure 136. TCD structure . . . . .	322
Figure 137. eDMA operation, part 1 . . . . .	331
Figure 138. eDMA operation, part 2 . . . . .	332
Figure 139. eDMA operation, part 3 . . . . .	333
Figure 140. Example of multiple loop iterations . . . . .	335
Figure 141. Memory array terms . . . . .	335
Figure 142. DMA_MUX block diagram . . . . .	345
Figure 143. Channel Configuration Registers (CHCONFIGn) . . . . .	347
Figure 144. DMA_MUX channel 0–3 block diagram . . . . .	351
Figure 145. DMA_MUX channel triggering: Normal operation . . . . .	351
Figure 146. DMA_MUX channel triggering: Ignored trigger . . . . .	352
Figure 147. DMA_MUX channel 4–15 block diagram . . . . .	353
Figure 148. INTC block diagram . . . . .	358
Figure 149. INTC Module Configuration Register (INTC_MCR) . . . . .	361
Figure 150. INTC Current Priority Register (INTC_CPR) . . . . .	362
Figure 151. INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 0 . . . . .	363
Figure 152. INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 1 . . . . .	363

Figure 153. INTC End-of-Interrupt Register (INTC_EOIR) . . . . .	364
Figure 154. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3]). . . . .	365
Figure 155. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7]). . . . .	365
Figure 156. INTC Priority Select Register 0–3 (INTC_PSR[0:3]). . . . .	366
Figure 157. INTC Priority Select Register 232-233 (INTC_PSR[232:233]) . . . . .	366
Figure 158. Software vector mode handshaking timing diagram . . . . .	379
Figure 159. Hardware vector mode handshaking timing diagram . . . . .	380
Figure 160. XBAR block diagram . . . . .	387
Figure 161. MPU block diagram . . . . .	393
Figure 162. MPU Control/Error Status Register (MPU_CESR) . . . . .	396
Figure 163. MPU Error Address Register, Slave Port n (MPU_EARn) . . . . .	397
Figure 164. MPU Error Detail Register, Slave Port n (MPU_EDRn) . . . . .	398
Figure 165. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0) . . . . .	399
Figure 166. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1) . . . . .	400
Figure 167. MPU Region Descriptor, Word 2 Register (MPU_RGDn.Word2) . . . . .	401
Figure 168. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3) . . . . .	404
Figure 169. MPU RGD Alternate Access Control n (MPU_RGDAACn). . . . .	405
Figure 170. MPU access evaluation macro . . . . .	407
Figure 171. System Integration Unit Lite block diagram . . . . .	412
Figure 172. MCU ID Register #1 (MIDR1) . . . . .	417
Figure 173. MCU ID Register #2 (MIDR2) . . . . .	418
Figure 174. Interrupt Status Flag Register (ISR) . . . . .	419
Figure 175. Interrupt Request Enable Register (IRER) . . . . .	420
Figure 176. Interrupt Rising-Edge Event Enable Register (IREER) . . . . .	420
Figure 177. Interrupt Falling-Edge Event Enable Register (IFEER) . . . . .	421
Figure 178. Interrupt Filter Enable Register (IFER) . . . . .	422
Figure 179. Pad Configuration Registers (PCRx) . . . . .	423
Figure 180. Pad Selection for Multiplexed Inputs Register (PSMIO_3) . . . . .	425
Figure 181. Port GPIO Pad Data Output Register 0–3 (GPDIO_3) . . . . .	430
Figure 182. Port GPIO Pad Data Input Register 0–3 (GPDIO_3) . . . . .	431
Figure 183. Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23) . . . . .	434
Figure 184. Interrupt Filter Clock Prescaler Register (IFCPR) . . . . .	435
Figure 185. Data Port example arrangement showing configuration for different port width accesses	436
Figure 186. External interrupt pad diagram . . . . .	437
Figure 187. I <sup>2</sup> C block diagram . . . . .	440
Figure 188. I <sup>2</sup> C Bus Address Register (IBAD) . . . . .	441
Figure 189. I <sup>2</sup> C Bus Frequency Divider Register (IBFD) . . . . .	442
Figure 190. SDA hold time . . . . .	443
Figure 191. SCL divider and SDA hold . . . . .	444
Figure 192. I <sup>2</sup> C Bus Control Register (IBCR) . . . . .	448
Figure 193. I <sup>2</sup> C Bus Status Register (IBSR) . . . . .	449
Figure 194. I <sup>2</sup> C Bus Data I/O Register (IBDR) . . . . .	450
Figure 195. I <sup>2</sup> C Bus Interrupt Config Register (IBIC) . . . . .	451
Figure 196. I <sup>2</sup> C module DMA interface block diagram . . . . .	452
Figure 197. I <sup>2</sup> C bus transmission signals . . . . .	453
Figure 198. Start and stop conditions . . . . .	454
Figure 199. I <sup>2</sup> C bus clock synchronization . . . . .	456
Figure 200. Flow-Chart of Typical I <sup>2</sup> C Interrupt Routine . . . . .	461
Figure 201. LIN topology network . . . . .	464
Figure 202. LINFlex block diagram . . . . .	464
Figure 203. LINFlex operating modes . . . . .	466
Figure 204. LINFlex in loop back mode . . . . .	467

Figure 205. LINFlex in self test mode	468
Figure 206. LIN control register 1 (LINCR1)	469
Figure 207. LIN interrupt enable register (LINIER)	472
Figure 208. LIN status register (LINSR)	474
Figure 209. LIN error status register (LINESR)	477
Figure 210. UART mode control register (UARTCR)	478
Figure 211. UART mode status register (UARTSR)	480
Figure 212. LIN timeout control status register (LINTCSR)	482
Figure 213. LIN output compare register (LINOOCR)	483
Figure 214. LIN timeout control register (LINTOCR)	483
Figure 215. LIN fractional baud rate register (LINFBR)	484
Figure 216. LIN integer baud rate register (LINIBRR)	485
Figure 217. LIN checksum field register (LINCFR)	486
Figure 218. LIN control register 2 (LINCR2)	486
Figure 219. Buffer identifier register (BIDR)	488
Figure 220. Buffer data register LSB (BDRL)	489
Figure 221. Buffer data register MSB (BDRM)	490
Figure 222. Identifier filter enable register (IFER)	490
Figure 223. Identifier filter match index (IFMI)	492
Figure 224. Identifier filter mode register (IFMR)	492
Figure 225. Identifier filter control register (IFCR2 $n$ )	494
Figure 226. Identifier filter control register (IFCR2 $n + 1$ )	495
Figure 227. UART mode 8-bit data frame	496
Figure 228. UART mode 9-bit data frame	496
Figure 229. Filter configuration—register organization	503
Figure 230. Identifier match index	504
Figure 231. LIN synch field measurement	505
Figure 232. Header and response timeout	507
Figure 233. LINFlexD block diagram	509
Figure 234. LIN network topology	511
Figure 235. LIN frame structure	512
Figure 236. Break field	512
Figure 237. Sync pattern	512
Figure 238. Structure of the data field	513
Figure 239. Identifier	513
Figure 240. LINFlexD controller operating modes	515
Figure 241. Filter configuration - register organization	522
Figure 242. Identifier match index	523
Figure 243. LIN sync field measurement	524
Figure 244. LINFlexD in Loop Back mode	525
Figure 245. LINFlexD in Self Test mode	526
Figure 246. UART mode 8-bit data frame	526
Figure 247. UART mode 9-bit data frame	527
Figure 248. UART mode 16-bit data frame	527
Figure 249. UART mode 17-bit data frame	527
Figure 250. LIN control register 1 (LINCR1)	533
Figure 251. LIN interrupt enable register (LINIER)	536
Figure 252. LIN status register (LINSR)	538
Figure 253. LIN error status register (LINESR)	541
Figure 254. UART mode control register (UARTCR)	542
Figure 255. UART mode status register (UARTSR)	545
Figure 256. LIN timeout control status register (LINTCSR)	547

Figure 257. LIN output compare register (LINOOCR) . . . . .	548
Figure 258. LIN timeout control register (LINTOCR) . . . . .	549
Figure 259. LIN timeout control register (LINTOCR) . . . . .	550
Figure 260. LIN integer baud rate register (LINIBRR) . . . . .	550
Figure 261. LIN checksum field register (LINCFR) . . . . .	551
Figure 262. LIN control register 2 (LINCR2) . . . . .	552
Figure 263. Buffer identifier register (BIDR) . . . . .	553
Figure 264. Buffer data register least significant (BDRL) . . . . .	554
Figure 265. Buffer data register most significant (BDRM) . . . . .	555
Figure 266. Identifier filter enable register (IFER) . . . . .	556
Figure 267. Identifier filter match index (IFMI) . . . . .	557
Figure 268. Identifier filter mode register (IFMR) . . . . .	558
Figure 269. Identifier filter control registers (IFCR0–IFCR15) . . . . .	559
Figure 270. Global control register (GCR) . . . . .	560
Figure 271. UART preset timeout register (UARTPTO) . . . . .	562
Figure 272. UART current timeout register (UARTCTO) . . . . .	562
Figure 273. DMA Tx enable register (DMATXE) . . . . .	563
Figure 274. DMA Rx enable register (DMARXE) . . . . .	564
Figure 275. TCD chain memory map (master node, TX mode) . . . . .	565
Figure 276. FSM to control the DMA TX interface (master node) . . . . .	566
Figure 277. TCD chain memory map (master node, RX mode) . . . . .	567
Figure 278. FSM to control the DMA RX interface (master node) . . . . .	568
Figure 279. TCD chain memory map (slave node, TX mode) . . . . .	569
Figure 280. FSM to control the DMA TX interface (slave node) . . . . .	571
Figure 281. TCD chain memory map (slave node, RX mode) . . . . .	572
Figure 282. FSM to control the DMA RX interface (slave node) . . . . .	574
Figure 283. TCD chain memory map (UART node, TX mode) . . . . .	575
Figure 284. FSM to control the DMA TX interface (UART node) . . . . .	576
Figure 285. TCD chain memory map (UART node, RX mode) . . . . .	577
Figure 286. FSM to control the DMA RX interface (UART node) . . . . .	579
Figure 287. Header and response timeout . . . . .	582
Figure 288. Interrupt diagram . . . . .	584
Figure 289. Programming consideration: master node, transmitter . . . . .	586
Figure 290. Programming consideration: master node, receiver . . . . .	586
Figure 291. Programming consideration: master node, transmitter, bit error . . . . .	586
Figure 292. Programming consideration: master node, receiver, checksum error . . . . .	586
Figure 293. Programming consideration: slave node, transmitter, no filters . . . . .	587
Figure 294. Programming consideration: slave node, receiver, no filters . . . . .	587
Figure 295. Programming consideration: slave node, transmitter, no filters, bit error . . . . .	587
Figure 296. Programming consideration: slave node, receiver, no filters, checksum error . . . . .	587
Figure 297. Programming consideration: slave node, at least one TX filter, BF is reset, ID matches filter . . . . .	588
Figure 298. Programming consideration: slave node, at least one RX filter, BF is reset, ID matches filter . . . . .	588
Figure 299. Programming consideration: slave node, RX only, TX only, RX and TX filters, ID not matching filter, BF is reset . . . . .	588
Figure 300. Programming consideration: slave node, TX filter, BF is set . . . . .	589
Figure 301. Programming consideration: slave node, RX filter, BF is set . . . . .	589
Figure 302. Programming consideration: slave node, TX filter, RX filter, BF is set . . . . .	590
Figure 303. Programming consideration: extended frames . . . . .	590
Figure 304. Programming consideration: response timeout . . . . .	591
Figure 305. Programming consideration: frame timeout . . . . .	591



Figure 306. Programming consideration: header timeout	591
Figure 307. Programming consideration: UART mode	591
Figure 308. FlexCAN block diagram	593
Figure 309. Message Buffer Structure	598
Figure 310. Rx FIFO structure	602
Figure 311. ID Table 0–7	602
Figure 312. Module Configuration Register (MCR)	604
Figure 313. Control Register (CTRL)	608
Figure 314. Free Running Timer (TIMER)	611
Figure 315. Rx Global Mask Register (RXGMASK)	613
Figure 316. Error Counter Register (ECR)	615
Figure 317. Error and Status Register (ESR)	616
Figure 318. Interrupt Masks 2 Register (IMASK2)	619
Figure 319. Interrupt Masks 1 Register (IMASK1)	620
Figure 320. Interrupt Flags 2 Register (IFLAG2)	621
Figure 321. Interrupt Flags 1 Register (IFLAG1)	622
Figure 322. Rx Individual Mask Registers (RXIMR0–RXIMR63)	623
Figure 323. CAN Engine Clocking Scheme	632
Figure 324. Segments within the Bit Time	633
Figure 325. Arbitration, Match and Move Time Windows	634
Figure 326. DSPI block diagram	639
Figure 327. DSPI with queues and eDMA	640
Figure 328. DSPI Module Configuration Register (DSPIx_MCR)	645
Figure 329. DSPI Transfer Count Register (DSPIx_TCR)	648
Figure 330. DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx_CTARn)	649
Figure 331. DSPI Status Register (DSPIx_SR)	656
Figure 332. DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)	658
Figure 333. DSPI PUSH TX FIFO Register (DSPIx_PUSHR)	660
Figure 334. DSPI POP RX FIFO Register (DSPIx_POPR)	662
Figure 335. DSPI Transmit FIFO Register 0–3 (DSPIx_TXFRn)	663
Figure 336. DSPI Receive FIFO Registers 0–3 (DSPIx_RXFRn)	664
Figure 337. SPI serial protocol overview	665
Figure 338. DSPI start and stop state diagram	666
Figure 339. Communications clock prescalers and scalars	670
Figure 340. Peripheral chip select strobe timing	672
Figure 341. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)	674
Figure 342. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8)	675
Figure 343. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} / 4$ )	677
Figure 344. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$ )	678
Figure 345. Example of non-continuous format (CPHA = 1, CONT = 0)	679
Figure 346. Example of continuous transfer (CPHA = 1, CONT = 1)	679
Figure 347. Polarity switching between frames	680
Figure 348. Continuous SCK timing diagram (CONT= 0)	681
Figure 349. Continuous SCK timing diagram (CONT=1)	682
Figure 350. TX FIFO pointers and counter	687
Figure 351. Interaction between timers and relevant peripherals	690
Figure 352. STM Control Register (STM_CR)	695
Figure 353. STM Count Register (STM_CNT)	695
Figure 354. STM Channel Control Register (STM_CCRn)	696
Figure 355. STM Channel Interrupt Register (STM_CIRn)	697
Figure 356. STM Channel Compare Register (STM_CMPn)	697
Figure 357. Channel configuration	700

Figure 358. eMIOS Module Configuration Register (EMIOSMCR) . . . . .	702
Figure 359. eMIOS Global FLAG (EMIOSGFLAG) Register . . . . .	704
Figure 360. eMIOS Output Update Disable (EMIOSOUDIS) Register. . . . .	704
Figure 361. eMIOS Enable Channel (EMIOSUCDIS) Register . . . . .	705
Figure 362. eMIOS UC A Register (EMIOSA[n]) . . . . .	706
Figure 363. eMIOS UC B Register (EMIOSB[n]) . . . . .	706
Figure 364. eMIOS UC Counter Register (EMIOSCNT[n]) . . . . .	707
Figure 365. eMIOS UC Control Register (EMIOSC[n]) . . . . .	708
Figure 366. eMIOS UC Status Register (EMIOSS[n]) . . . . .	712
Figure 367. eMIOS UC Alternate A register (EMIOSALTA[n]). . . . .	713
Figure 368. Single action input capture with rising edge triggering example. . . . .	715
Figure 369. Single action input capture with both edges triggering example. . . . .	716
Figure 370. SAOC example with EDPOL value being transferred to the output flip-flop . . . . .	717
Figure 371. SAOC example toggling the output flip-flop . . . . .	717
Figure 372. SAOC example with flag behavior . . . . .	717
Figure 373. Input pulse width measurement example . . . . .	718
Figure 374. B1 and A1 updates at EMIOSA[n] and EMIOSB[n] reads . . . . .	719
Figure 375. Input period measurement example . . . . .	720
Figure 376. A1 and B1 updates at EMIOSA[n] and EMIOSB[n] reads . . . . .	720
Figure 377. Double action output compare with FLAG set on the second match . . . . .	721
Figure 378. Double action output compare with FLAG set on both matches. . . . .	722
Figure 379. DAOC with transfer disabling example . . . . .	722
Figure 380. Modulus Counter Up mode example . . . . .	724
Figure 381. Modulus Counter Up/Down mode example . . . . .	724
Figure 382. Modulus Counter Buffered (MCB) Up Count mode . . . . .	725
Figure 383. Modulus Counter Buffered (MCB) Up/Down mode. . . . .	726
Figure 384. MCB Mode A1 Register Update in Up Counter mode . . . . .	726
Figure 385. MCB Mode A1 Register Update in Up/Down Counter mode . . . . .	727
Figure 386. OPWFMB A1 and B1 match to Output Register Delay. . . . .	728
Figure 387. OPWFMB Mode with A1 = 0 (0% duty cycle). . . . .	729
Figure 388. OPWFMB A1 and B1 registers update and flags . . . . .	730
Figure 389. OPWFMB mode from 100% to 0% duty cycle . . . . .	730
Figure 390. OPWMCB A1 and B1 registers load. . . . .	732
Figure 391. OPWMCB with lead dead time insertion. . . . .	733
Figure 392. OPWMCB with trail dead time insertion . . . . .	734
Figure 393. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3). . . . .	735
Figure 394. OPWMB mode matches and flags . . . . .	737
Figure 395. OPWMB mode with 0% duty cycle . . . . .	738
Figure 396. OPWMB mode from 100% to 0% duty cycle . . . . .	738
Figure 397. OPWMT example . . . . .	741
Figure 398. OPWMT with 0 % Duty Cycle . . . . .	741
Figure 399. OPWMT with 100% duty cycle . . . . .	742
Figure 400. Input programmable filter submodule diagram . . . . .	742
Figure 401. Input programmable filter example . . . . .	743
Figure 402. Time base period when running in the fastest prescaler ratio . . . . .	745
Figure 403. Time base generation with external clock and clear on match start. . . . .	746
Figure 404. Time base generation with internal clock and clear on match start . . . . .	746
Figure 405. Time base generation with clear on match end . . . . .	747
Figure 406. PIT block diagram. . . . .	748
Figure 407. PIT Module Control Register (PITMCR) . . . . .	750
Figure 408. Timer Load Value Register (LDVAL) . . . . .	751
Figure 409. Current Timer Value Register (CVAL) . . . . .	751

Figure 410. Timer Control Register (TCTRL) . . . . .	752
Figure 411. Timer Flag Register (TFLG) . . . . .	753
Figure 412. Stopping and starting a timer . . . . .	754
Figure 413. Modifying running timer period . . . . .	754
Figure 414. Dynamically setting a new load value . . . . .	754
Figure 415. Implementation of ADC_0 and ADC_1 . . . . .	757
Figure 416. Normal conversion flow . . . . .	759
Figure 417. Injected sample/conversion sequence . . . . .	760
Figure 418. Sampling and conversion timings . . . . .	762
Figure 419. Sampling and conversion timings . . . . .	763
Figure 420. Presampling sequence . . . . .	767
Figure 421. Presampling sequence with PRECONV = 1 . . . . .	767
Figure 422. Guarded area . . . . .	768
Figure 423. Main Configuration Register (MCR) . . . . .	778
Figure 424. Main Status Register (MSR) . . . . .	780
Figure 425. Interrupt Status Register (ISR) . . . . .	781
Figure 426. Channel Pending Register 0 (CEO CFR0) . . . . .	782
Figure 427. Channel Pending Register 1 (CEO CFR1) - ADC_0 . . . . .	782
Figure 428. Channel Pending Register 1 (CEO CFR1) - ADC_1 . . . . .	783
Figure 429. Channel Pending Register 2 (CEO CFR2) . . . . .	783
Figure 430. Interrupt Mask Register (IMR) . . . . .	784
Figure 431. Channel Interrupt Mask Register 0 (CIMR0) . . . . .	785
Figure 432. Channel Interrupt Mask Register 1 (CIMR1) - ADC_0 . . . . .	785
Figure 433. Channel Interrupt Mask Register 1 (CIMR1) - ADC_1 . . . . .	785
Figure 434. Channel Interrupt Mask Register 2 (CIMR2) . . . . .	786
Figure 435. ADC_0 Watchdog Threshold Interrupt Status Register (WTISR) . . . . .	786
Figure 436. ADC_1 Watchdog Threshold Interrupt Status Register (WTISR) . . . . .	787
Figure 437. ADC_0 Watchdog Threshold Interrupt Mask Register (WTIMR) . . . . .	787
Figure 438. ADC_1 Watchdog Threshold Interrupt Mask Register (WTIMR) . . . . .	788
Figure 439. DMA Enable Register (DMAE) . . . . .	789
Figure 440. DMA Channel Select Register 0 (DMAR0) . . . . .	790
Figure 441. DMA Channel Select Register 1 (DMAR1) - ADC_0 . . . . .	790
Figure 442. DMA Channel Select Register 1 (DMAR1) - ADC_1 . . . . .	790
Figure 443. DMA Channel Select Register 2 (DMAR2) . . . . .	791
Figure 444. ADC_0 Threshold Register THRHLR[0...5] . . . . .	791
Figure 445. ADC_1 Threshold Register THRHLR[0...2] . . . . .	792
Figure 446. Presampling Control Register (PSCR) . . . . .	793
Figure 447. Presampling Register 0 (PSR0) . . . . .	794
Figure 448. Presampling Register 1 (PSR1) - ADC_0 . . . . .	794
Figure 449. Presampling Register 1 (PSR1) - ADC_1 . . . . .	794
Figure 450. Presampling Register 2 (PSR2) . . . . .	795
Figure 451. Conversion timing registers CTR[0...2] . . . . .	796
Figure 452. Normal Conversion Mask Register 0 (NCFMR0) . . . . .	797
Figure 453. Normal Conversion Mask Register 1 (NCFMR1) - ADC_0 . . . . .	797
Figure 454. Normal Conversion Mask Register 1 (NCFMR1) - ADC_1 . . . . .	798
Figure 455. Normal Conversion Mask Register 2 (NCFMR2) . . . . .	798
Figure 456. Injected Conversion Mask Register 0 (JCFMR0) . . . . .	799
Figure 457. Injected Conversion Mask Register 1 (JCFMR1) - ADC_0 . . . . .	799
Figure 458. Injected Conversion Mask Register 1 (JCFMR1) - ADC_1 . . . . .	800
Figure 459. Injected Conversion Mask Register 2 (JCFMR2) . . . . .	800
Figure 460. Decode Signals Delay Register (DSDR) . . . . .	801
Figure 461. Power-down Exit Delay Register (PDED R) . . . . .	802

Figure 462. Channel Data Register (CDR[0...95]) . . . . .	803
Figure 463. Channel Data Register (CDR[0...95]) . . . . .	803
Figure 464. Channel Watchdog Select Register 0 (CWSELR0) – ADC_0. . . . .	805
Figure 465. Channel Watchdog Select Register 1 (CWSELR1) – ADC_0. . . . .	805
Figure 466. Channel Watchdog Select Register 4 (CWSELR4) – ADC_0. . . . .	806
Figure 467. Channel Watchdog Select Register 5 (CWSELR5) – ADC_0. . . . .	806
Figure 468. Channel Watchdog Select Register 6 (CWSELR6) – ADC_0. . . . .	807
Figure 469. Channel Watchdog Select Register 7 (CWSELR7) – ADC_0. . . . .	807
Figure 470. Channel Watchdog Select Register 8 (CWSELR8) – ADC_0. . . . .	808
Figure 471. Channel Watchdog Select Register 9 (CWSELR9) – ADC_0. . . . .	808
Figure 472. Channel Watchdog Select Register 10 (CWSELR10) – ADC_0. . . . .	809
Figure 473. Channel Watchdog Select Register 11 (CWSELR11) – ADC_0. . . . .	809
Figure 474. Channel Watchdog Select Register 0 (CWSELR0) – ADC_1. . . . .	810
Figure 475. Channel Watchdog Select Register 1 (CWSELR1) – ADC_1. . . . .	810
Figure 476. Channel Watchdog Select Register 4 (CWSELR4) – ADC_1. . . . .	811
Figure 477. Channel Watchdog Select Register 5 (CWSELR5) – ADC_1. . . . .	811
Figure 478. Channel Watchdog Enable Register 0 (CWENR0). . . . .	812
Figure 479. Channel Watchdog Enable Register 1 (CWENR1). . . . .	812
Figure 480. Channel Watchdog Enable Register 1 (CWENR1) - ADC_1 . . . . .	813
Figure 481. Channel Watchdog Enable Register 2 (CWENR2). . . . .	813
Figure 482. Analog Watchdog Out of Range Register 0 (AWORR0). . . . .	814
Figure 483. Analog Watchdog Out of Range Register 1 (AWORR1) - ADC_0 . . . . .	814
Figure 484. Analog Watchdog Out of Range Register 1 (AWORR1) - ADC_1 . . . . .	815
Figure 485. Analog Watchdog Out of Range Register 2 (AWORR2). . . . .	815
Figure 486. Cross Triggering Unit block diagram . . . . .	816
Figure 487. Event Configuration Registers (CTU_EVTCFGRx) (x = 0...63) . . . . .	817
Figure 488. Flash memory architecture . . . . .	823
Figure 489. CFlash and DFlash module structures . . . . .	825
Figure 490. CFlash Module Configuration Register (CFLASH_MCR) . . . . .	834
Figure 491. CFlash Low/Mid Address Space Block Locking Register (CFLASH_LML) . . . . .	839
Figure 492. CFlash Nonvolatile Low/Mid address space block Locking register (CFLASH_NVLML) . . . . .	842
Figure 493. CFlash Nonvolatile High Address Space Block Locking Register (CFLASH_HBL) . . . . .	844
Figure 494. CFlash Nonvolatile High Address Space Block Locking Register (CFLASH_NVHBL) . . . . .	845
Figure 495. CFlash Secondary Low/mid address space block Locking Register (CFLASH_SLL) . . . . .	847
Figure 496. CFlash Nonvolatile Secondary Low/mid address space block Locking register (CFLASH_N VSL) . . . . .	850
Figure 497. CFlash Low/Mid address space block Select register (CFLASH_LMS) . . . . .	853
Figure 498. CFlash High Address Space Block Select Register (CFLASH_HBS). . . . .	855
Figure 499. CFlash Address Register (CFLASH_ADR). . . . .	856
Figure 500. CFlash User Test 0 register (CFLASH_UT0) . . . . .	857
Figure 501. CFlash User Test 1 register (CFLASH_UT1) . . . . .	859
Figure 502. CFlash User Test 2 register (CFLASH_UT2) . . . . .	860
Figure 503. CFlash User Multiple Input Signature Register 0 (CFLASH_UMISR0). . . . .	861
Figure 504. CFlash User Multiple Input Signature Register 1 (CFLASH_UMISR1). . . . .	862
Figure 505. CFlash User Multiple Input Signature Register 2 (CFLASH_UMISR2). . . . .	863
Figure 506. CFlash User Multiple Input Signature Register 3 (CFLASH_UMISR3). . . . .	864
Figure 507. CFlash User Multiple Input Signature Register 4 (CFLASH_UMISR4). . . . .	865
Figure 508. CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0) . . . . .	866
Figure 509. CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1) . . . . .	866
Figure 510. CFlash Nonvolatile System Censorship Control 0 register (NVSCC0) . . . . .	867
Figure 511. CFlash Nonvolatile System Censorship Control 1 register (NVSCC1) . . . . .	868
Figure 512. CFlash Nonvolatile User Options register (NVUSRO) . . . . .	869

Figure 513. DFlash Module Configuration Register (DFLASH_MCR) . . . . .	870
Figure 514. DFlash Low/Mid Address Space Block Locking Register (DFLASH_LML) . . . . .	875
Figure 515. DFlash Nonvolatile Low/Mid address space block Locking register (DFLASH_NVLML) . . . . .	877
Figure 516. DFlash Secondary Low/mid address space block Locking register (DFLASH_SLL) . . . . .	879
Figure 517. DFlash Nonvolatile Secondary Low/mid address space block Locking register (DFLASH_N VSLL) . . . . .	881
Figure 518. DFlash Low/Mid Address Space Block Select Register (DFLASH_LMS) . . . . .	883
Figure 519. DFlash Address Register (DFLASH_ADR) . . . . .	884
Figure 520. DFlash User Test 0 register (DFLASH_UT0) . . . . .	885
Figure 521. DFlash User Test 1 register (DFLASH_UT1) . . . . .	887
Figure 522. DFlash User Test 2 register (DFLASH_UT2) . . . . .	888
Figure 523. DFlash User Multiple Input Signature Register 0 (DFLASH_UMISR0) . . . . .	889
Figure 524. DFlash User Multiple Input Signature Register 1 (DFLASH_UMISR1) . . . . .	890
Figure 525. DFlash User Multiple Input Signature Register 2 (DFLASH_UMISR2) . . . . .	891
Figure 526. DFlash User Multiple Input Signature Register 3 (DFLASH_UMISR3) . . . . .	892
Figure 527. DFlash User Multiple Input Signature Register 4 (DFLASH_UMISR4) . . . . .	893
Figure 528. Power Architecture e200z0h RPP reference platform block diagram . . . . .	905
Figure 529. PFlash Configuration Register 0 (PFCR0) . . . . .	909
Figure 530. PFlash Configuration Register 1 (PFCR1) . . . . .	912
Figure 531. PFlash Access Protection Register (PFAPR) . . . . .	915
Figure 532. Nonvolatile Platform Flash Access Protection Register (NVPFAPR) . . . . .	916
Figure 533. Register Protection block diagram . . . . .	928
Figure 534. Register protection memory diagram . . . . .	929
Figure 535. Soft Lock Bit Register (SLBRn) . . . . .	931
Figure 536. Global Configuration Register (GCR) . . . . .	932
Figure 537. Change Lock Settings Directly Via Area #4 . . . . .	934
Figure 538. Change Lock Settings for 16-bit Protected Addresses . . . . .	934
Figure 539. Change Lock Settings for 32-bit Protected Addresses . . . . .	935
Figure 540. Change Lock Settings for Mixed Protection . . . . .	935
Figure 541. Enable Locking Via Mirror Module Space (Area #3) . . . . .	936
Figure 542. Enable Locking for Protected and Unprotected Addresses . . . . .	936
Figure 543. SWT Control Register (SWT_CR) . . . . .	946
Figure 544. SWT Interrupt Register (SWT_IR) . . . . .	947
Figure 545. SWT Time-Out Register (SWT_TO) . . . . .	948
Figure 546. SWT Window Register (SWT_WN) . . . . .	948
Figure 547. SWT Service Register (SWT_SR) . . . . .	949
Figure 548. SWT Counter Output Register (SWT_CO) . . . . .	949
Figure 549. Processor Core Type Register (PCT) . . . . .	954
Figure 550. SoC-Defined Platform Revision Register (REV) . . . . .	954
Figure 551. IPS On-Platform Module Configuration Register (IOPMC) . . . . .	955
Figure 552. Miscellaneous Wakeup Control (MWCR) Register . . . . .	956
Figure 553. Miscellaneous Interrupt (MIR) Register . . . . .	957
Figure 554. Miscellaneous User-Defined Control (MUDCR) Register . . . . .	958
Figure 555. ECC Configuration (ECR) Register . . . . .	959
Figure 556. ECC Status Register (ESR) . . . . .	962
Figure 557. ECC Error Generation Register (EEGR) . . . . .	963
Figure 558. Platform Flash ECC Address Register (PFEAR) . . . . .	966
Figure 559. Platform Flash ECC Master Number Register (PFEMR) . . . . .	966
Figure 560. Platform Flash ECC Attributes Register (PFEAT) . . . . .	967
Figure 561. Platform Flash ECC Data Register (PFEDR) . . . . .	968
Figure 562. Platform RAM ECC Address Register (PREAR) . . . . .	968
Figure 563. Platform RAM ECC Syndrome Register (PRESR) . . . . .	969

Figure 564. Platform RAM ECC Master Number Register (PREMR) . . . . .	971
Figure 565. Platform RAM ECC Attributes Register (PREAT) . . . . .	972
Figure 566. Platform RAM ECC Data Register (PREDR) . . . . .	973
Figure 567. JTAG Controller Block Diagram . . . . .	974
Figure 568. 5-bit Instruction Register . . . . .	977
Figure 569. Device Identification Register . . . . .	977
Figure 570. Shifting data through a register . . . . .	978
Figure 571. IEEE 1149.1-2001 TAP controller finite state machine . . . . .	979
Figure 572. e200z0 OnCE Block Diagram . . . . .	983
Figure 573. OnCE Command Register (OCMD) . . . . .	984
Figure 574. NDI functional block diagram . . . . .	987
Figure 575. NDI implementation block diagram . . . . .	988
Figure 576. 4-Bit Instruction Register . . . . .	996
Figure 577. Port Configuration Register (PCR) . . . . .	997
Figure 578. MSEO Transfers . . . . .	999
Figure 579. Message Field Sizes . . . . .	1000
Figure 580. Transmission Sequence of Messages . . . . .	1000
Figure 581. NEXUS Controller State Machine . . . . .	1001
Figure 582. IEEE, 1149.1 Controller Command Input . . . . .	1002
Figure 583. e200z0 Nexus2+ Functional Block Diagram . . . . .	1004
Figure 584. Development Control Register 1 (DC1) . . . . .	1009
Figure 585. Development Control Register 2 (DC2) . . . . .	1010
Figure 586. Development Status Register (DS) . . . . .	1011
Figure 587. Read/Write Access Control/Status Register (RWCS) . . . . .	1012
Figure 588. Read/Write Access Address Register (RWA) . . . . .	1013
Figure 589. Read/Write Access Data Register (RWD) . . . . .	1014
Figure 590. Watchpoint Trigger Register (WT) . . . . .	1015

# 1 Preface

## 1.1 Overview

The primary objective of this document is to define the functionality of the SPC560B54/6x microcontroller for use by software and hardware developers. The SPC560B54/6x is built on Power Architecture® technology and integrates technologies that are important for today's automotive vehicle body applications.

The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, it is the reader's responsibility to be sure he or she is using the most recent version of the documentation.

To locate any published errata or updates for this document, visit the ST Web site at [www.st.com](http://www.st.com).

## 1.2 Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the SPC560B54/6x device. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture.

## 1.3 Guide to this reference manual

Table 1. Guide to this reference manual

Chapter		Description	Functional group
#	Title		
2	<i>Introduction</i>	General overview, family description, feature list and information on how to use the reference manual in conjunction with other available documents.	Introductory material
3	<i>Memory Map</i>	Memory map of all peripherals and memory.	Memory map
4	<i>Signal description</i>	Pinout diagrams and descriptions of all pads.	Signals
5	<i>Microcontroller Boot</i>		Boot
	<i>– Boot mechanism</i>	– Describes what configuration is required by the user and what processes are involved when the microcontroller boots from flash memory or serial boot modes. – Describes censorship.	
	<i>– Boot Assist Module (BAM)</i>	Features of BAM code and when it's used.	
	<i>– System Status and Configuration Module (SSCM)</i>	Reports information about current state and configuration of the microcontroller.	

**Table 1. Guide to this reference manual (continued)**

Chapter		Description	Functional group
#	Title		
6	<i>Clock Description</i>	<ul style="list-style-type: none"> <li>– Covers configuration of all of the clock sources in the system.</li> <li>– Describes the Clock Monitor Unit (CMU).</li> </ul>	Clocks and power  (includes operating mode configuration and how to wake up from low power mode)
7	<i>Clock Generation Module (MC_CGM)</i>	Determines how the clock sources are used (including clock dividers) to generate the reference clocks for all of the modules and peripherals.	
8	<i>Mode Entry Module (MC_ME)</i>	Determines the clock source, memory, power and peripherals that are available in each operating mode.	
9	<i>Reset Generation Module (MC_RGM)</i>	Manages the process of entering and exiting reset, allows reset sources to be configured (including LVD's) and provides status reporting.	
10	<i>Power Control Unit (MC_PCU)</i>	Controls the power to different power domains within the microcontroller (allowing SRAM to be selectively powered in STANDBY mode).	
11	<i>Voltage Regulators and Power Supplies</i>	Information on voltage regulator implementation. Includes enable bit for 5 V LVD (see also MC_RGM).	
12	<i>Wakeup Unit (WKPU)</i>	Always-active analog block. Details configuration of 2 internal (API/RTC) and 27 external (pin) low power mode wakeup sources.	
13	<i>Real Time Clock / Autonomous Periodic Interrupt (RTC/API)</i>	Details configuration and operation of timers that are predominately used for system wakeup.	
14	<i>CAN Sampler</i>	Details on how to configure the CAN sampler which is used to capture the identifier frame of a CAN message when the microcontroller is in low power mode.	



**Table 1. Guide to this reference manual (continued)**

Chapter		Description	Functional group
#	Title		
15	<a href="#">e200z0h Core</a>	Overview on cores. For more details consult the core reference manuals available on <a href="http://www.st.com">www.st.com</a> .	Core platform modules
16	<a href="#">Enhanced Direct Memory Access (eDMA)</a>	Operation and configuration information on the 32-channel direct memory access that can be used to transfer data between any memory mapped locations. Certain peripherals have eDMA triggers that can be used to feed configuration data to, or read results from the peripherals.	
17	<a href="#">eDMA Channel Multiplexer (DMA_MUX)</a>	Operation and configuration information for the eDMA multiplexer, which takes the 59 possible eDMA sources (triggers from the DSPI, eMIOS, I <sup>2</sup> C, ADC and LINFlexD) and multiplexes them onto the 16 eDMA channels.	
18	<a href="#">Interrupt Controller (INTC)</a>	Provides the configuration and control of all of the external interrupts (non-core) that are then routed to the IVOR4 core interrupt vector.	
19	<a href="#">Crossbar Switch (XBAR)</a>	Describes the connections of the XBAR masters and slaves on this microcontroller.	
20	<a href="#">Memory Protection Unit (MPU)</a>	The MPU sits on the slave side of the XBAR and allows highly configurable control over all master accesses to the memory.	
21	<a href="#">System Integration Unit Lite (SIUL)</a>	How to configure the pins or ports for input or output functions including external interrupts and DSI serialization.	Ports
22	<a href="#">Inter-Integrated Circuit Bus Controller Module (I2C)</a>	These chapters describe the configuration and operation of the various communication modules. Some of these modules support DMA requests to fill / empty buffer queues to minimize CPU overhead.	Communication modules
23	<a href="#">LIN Controller (LINFlex)</a>		
24	<a href="#">LIN Controller (LINFlexD)</a>		
25	<a href="#">FlexCAN</a>		
26	<a href="#">Deserial Serial Peripheral Interface (DSPI)</a>		

**Table 1. Guide to this reference manual (continued)**

Chapter		Description	Functional group
#	Title		
27	<i>Timers</i>		Timer modules
	<i>– Technical overview</i>	Gives an overview of the available system timer modules showing links to other modules as well as tables detailing the external pins associated with eMIOS timer channels.	
	<i>– System Timer Module (STM)</i>	A simple 32-bit free running counter with 4 compare channels with interrupt on match. It can be read at any time; this is very useful for measuring execution times.	
	<i>– Enhanced Modular IO Subsystem (eMIOS)</i>	Highly configurable timer module(s) supporting PWM, output compare and input capture features. Includes interrupt and eDMA support.	
	<i>– Periodic Interrupt Timer (PIT)</i>	Set of 32-bit countdown timers that provide periodic events (which can trigger an interrupt) with automatic re-load.	
28	<i>Analog-to-Digital Converter (ADC)</i>	Details the configuration and operation of the ADC modules as well as detailing the channels that are shared between the 10-bit and 12-bit ADC. The ADC is tightly linked to the INTC, eDMA, PIT, and CTU. When used in conjunction with these other modules, the CPU overhead for an ADC conversion is significantly reduced.	ADC system
29	<i>Cross Triggering Unit (CTU)</i>	The CTU allows an ADC conversion to be automatically triggered based on an eMIOS event (like a PWM output going high) or a PIT_RTI event with no CPU intervention.	
30	<i>Flash Memory</i>	Details the code and data flash memory structure (with ECC), block sizes and the flash memory port configuration, including wait states, line buffer configuration and pre-fetch control.	Memory
31	<i>Static RAM (SRAM)</i>	Details the structure of the SRAM (with ECC). There are no user configurable registers associated with the SRAM.	

**Table 1. Guide to this reference manual (continued)**

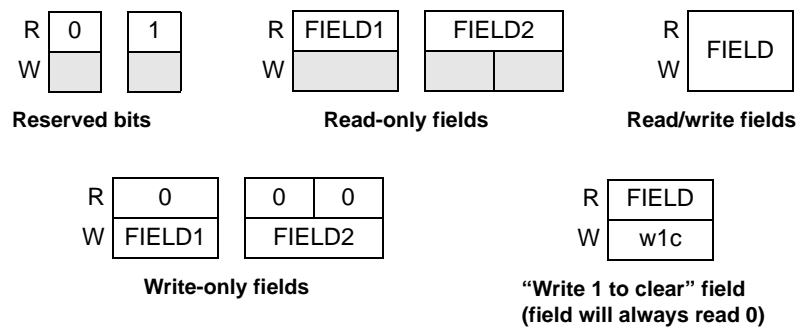
Chapter		Description	Functional group
#	Title		
32	<i>Register Protection</i>	Certain registers in each peripheral can be protected from further writes using the register protection mechanism detailed in this section. Registers can either be configured to be unlocked via a soft lock bit or locked until the next reset.	Integrity
33	<i>Software Watchdog Timer (SWT)</i>	The SWT offers a selection of configurable modes that can be used to monitor the operation of the microcontroller and /or reset the device or trigger an interrupt if the SWT is not correctly serviced. The SWT is enabled out of reset.	
34	<i>Error Correction Status Module (ECSM)</i>	Provides information about the last reset, general device information, system fault information and detailed ECC error information.	
35	<i>IEEE 1149.1 Test Access Port Controller (JTAGC)</i>	Used for boundary scan as well as device debug.	Debug
36	<i>Nexus Development Interface (NDI)</i>	Provides advanced debug features including non intrusive trace capabilities.	
37	<i>Revision History</i>	Summarizes the changes between each successive revision of this reference manual	Revision history information

## 1.4 Register description conventions

The register information for SPC560B54/6x is presented in:

- Memory maps containing:
  - An offset from the module’s base address
  - The name and acronym/abbreviation of each register
  - The page number on which each register is described
- Register figures
- Field-description tables
- Associated text

The register figures show the field structure using the conventions in *Figure 1*.



**Figure 1. Register figure conventions**

The numbering of register bits and fields on SPC560B54/6x is as follows:

- Register bit numbers, shown at the top of each figure, use the standard Power Architecture bit ordering (0, 1, 2, ...) where bit 0 is the most significant bit (MSB).
- Multi-bit fields within a register use conventional bit ordering (... , 2, 1, 0) where bit 0 is the least significant bit (LSB).

## 1.5 References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC560B54/6x:

- IEEE-ISTO 5001-2003 Standard for a Global Embedded Processor Interface (Nexus)
- IEEE 1149.1-2001 standard - IEEE Standard Test Access Port and Boundary-Scan Architecture

## 1.6 How to use the SPC560B54/6x documents

This section:

- Describes how the SPC560B54/6x documents provide information on the microcontroller
- Makes recommendations on how to use the documents in a system design

### 1.6.1 The SPC560B54/6x document set

The SPC560B54/6x document set comprises:

- This reference manual (provides information on the features of the logical blocks on the device and how they are integrated with each other)
- The device data sheet (specifies the electrical characteristics of the device)
- The device product brief

The following reference documents (available online at [www.st.com](http://www.st.com)) are also available to support the CPU on this device:

- *Programmer's Reference Manual for Book E Processors*
- Variable-Length Encoding (VLE) Extension - Programming Interface Manual

The aforementioned documents describe all of the functional and electrical characteristics of the SPC560B54/6x microcontroller.

Depending on your task, you may need to refer to multiple documents to make design decisions. However, in general the use of the documents can be divided up as follows:

- Use the reference manual (this document) during software development and when allocating functions during system design.
- Use the data sheet when designing hardware and optimizing power consumption.
- Use the CPU reference documents when:
  - Configuring CPU memory and branch optimizations
  - Doing detailed software development in assembly language
  - Debugging complex software interactions

## 1.6.2 Reference manual content

The content in this document focuses on the functionality of the microcontroller rather than its performance. Most chapters describe the functionality of a particular on-chip module, such as a CAN controller or timer. The remaining chapters describe how these modules are integrated into the memory map, how they are powered and clocked, and the pin-out of the device.

In general, when an individual module is enabled for use all of the detail required to configure and operate it is contained in the dedicated chapter. In some cases there are multiple implementations of this module, however, there is only one chapter for each type of module in use. For this reason, the address of registers in each module is normally provided as an offset from a base address which can be found in [Chapter 3: Memory Map](#). The benefit of this approach is that software developed for a particular module can be easily reused on this device and on other related devices that use the same modules.

The steps to enable a module for use varies but typically these require configuration of the integration features of the microcontroller. The module will normally have to be powered and enabled at system level, then a clock may have to be explicitly chosen and finally if required the input and output connections to the external system must be configured.

The primary integration chapters of the reference manual contain most of the information required to enable the modules. There are special cases where a chapter may describe module functionality and some integration features for convenience — for example, the microcontroller input/output (SIUL) module. Integration and functional content is provided in the manual as shown in [Table 2](#).

**Table 2. Reference manual integration and functional content**

Chapter	Integration content	Functional content
Introduction	<ul style="list-style-type: none"> <li>– The main features on chip</li> <li>– A summary of the functions provided by each module</li> </ul>	—
Memory Map	How the memory map is allocated, including: <ul style="list-style-type: none"> <li>– Internal RAM</li> <li>– Flash memory</li> <li>– External memory-mapped resources and the location of the registers used by the peripherals<sup>(1)</sup></li> </ul>	—
Signal Description	How the signals from each of the modules are combined and brought to a particular pin on a package	—
Boot Assist Module	CPU boot sequence from reset Implementation of the boot options if internal flash memory is not used	
Clock Description	Clocking architecture of the device (which clock is available for the system and each peripheral)	Description of operation of different clock sources
eDMA Channel Multiplexer	Source values for module eDMA channels	How to connect a module eDMA channel to the eDMA module
Interrupt Controller	Interrupt vector table	Operation of the module
Mode Entry Module	Module numbering for control and status	Operation of operating modes
System Integration Unit Lite	How input signals are mapped to individual modules including external interrupt pins	Operation of GPIO
Voltage regulators and power supplies	Power distribution to the MCU	—
Wakeup Unit	Allocation of inputs to the Wakeup Unit	Operation of the wakeup feature

1. To find the address of a register in a particular module take the start address of the module given in the memory map and add the offset for the register given in the module chapter.

## 1.7 Using the SPC560B54/6x

There are many different approaches to designing a system using the SPC560B54/6x so the guidance in this section is provided as an example of how the documents can be applied in this task.

Familiarity with the SPC560B54/6x modules can help ensure that its features are being optimally used in a system design. Therefore, the current chapter is a good starting point. Further information on the detailed features of a module are provided within the module chapters. These, combined with the current chapter, should provide a good introduction to the functions available on the MCU.

## 1.7.1 Hardware design

The SPC560B54/6x requires that certain pins are connected to particular power supplies, system functions and other voltage levels for operation.

The SPC560B54/6x internal logic operates from 1.2 V (nominal) supplies that are normally supplied by the on-chip voltage regulator from a 5 V or 3.3 V supply. The 3.3–5 V ( $\pm 10\%$ ) supply is also used to supply the input/output pins on the MCU. [Chapter 4: Signal description](#), describes the power supply pin names, numbers and their purpose. For more detail on the voltage supply of each pin, see [Chapter 11: Voltage Regulators and Power Supplies](#). For specifications of the voltage ranges and limits and decoupling of the power supplies see the SPC560B54/6x data sheet.

Certain pins have dedicated functions that affect the behavior of the MCU after reset. These include pins to force test or alternate boot conditions and debug features. These are described in [Chapter 4: Signal description](#), and a hardware designer should take care that these pins are connected to allow correct operation.

Beyond power supply and pins that have special functions there are also pins that have special system purposes such as oscillator and reset pins. These are also described in [Chapter 4: Signal description](#). The reset pin is bidirectional and its function is closely tied to the reset generation module [[Chapter 9: Reset Generation Module \(MC\\_RGM\)](#)]. The crystal oscillator pins are dedicated to this function but the oscillator is not started automatically after reset. The oscillator module is described in [Section 6.3: Fast external crystal oscillator \(FXOSC\) digital interface](#), along with the internal clock architecture and the other oscillator sources on chip.

## 1.7.2 Input/output pins

The majority of the pins on the MCU are input/output pins which may either operate as general purpose pins or be connected to a particular on-chip module. The arrangement allows a function to be available on several pins. The system designer should allocate the function for the pin before connecting to external hardware. The software should then choose the correct function to match the hardware. The pad characteristics can vary depending on the functions on the pad. [Chapter 4: Signal description](#), describes each pad type (for example, S, M, or J). Two pads may be able to carry the same function but have different pad types. The electrical specification of the pads is described in the data sheet dependent on the function enabled and the pad type.

There are three modules that configure the various functions available:

- System Integration Unit Lite (SIUL)
- Wakeup Unit (WKPU)
- 32 KHz oscillator (SXOSC)

The SIUL configures the digital pin functions. Each pin has a register (PCR) in the module that allows selection of the output functions that is connected to the pin. The available settings for the PCR are described in [Section 4.7: Functional port pins](#). Inputs are selected using the PSMI registers; these are described in [Chapter 21: System Integration Unit Lite \(SIUL\)](#). (PSMI registers connect a module to one of several pins, whereas the PCR registers connect a pin to one of several modules).

The WKPU provides the ability to cause interrupts and wake the MCU from low power modes and operates independently from the SIUL.

In addition to digital I/O functions, the SXOSC is a "special function" that provides a slow external crystal. The SXOSC is enabled independently from the digital I/O which means that the digital function on the pin must be disabled when the SXOSC is active.

The ADC functions are enabled using the PCRs.

### 1.7.3 Software design

Certain modules provide system integration functions, and other modules (such as timers) provide specific functions.

From reset, the modules involved in configuring the system for application software are:

- Boot Assist Module (BAM) — determines the selected boot source
- Reset Generation Module (MC\_RGM) — determines the behavior of the MCU when various reset sources are triggered and reports the source of the reset
- Mode Entry Module (MC\_ME) — controls which operating mode the MCU is in and configures the peripherals and clocks and power supplies for each of the modes
- Power Control Unit (MC\_PCU) — determines which power domains are active
- Clock Generation Module (MC\_CGM) — chooses the clock source for the system and many peripherals

After reset, the MCU will automatically select the appropriate reset source and begin to execute code. At this point the system clock is the 16 MHz FIRC oscillator, the CPU is in supervisor mode and all the memory is available. Initialization is required before most peripherals may be used and before the SRAM can be read (since the SRAM is protected by ECC, the syndrome will generally be uninitialized after reset and reads would fail the check). Accessing disabled features causes error conditions or interrupts.

A typical startup routine would involve initializing the software environment including stacks, heaps, variable initialization and so on and configuring the MCU for the application.

The MC\_ME module enables the modules and other features like clocks. It is therefore an essential part of the initialization and operation software. In general, the software will configure an MC\_ME mode to make certain peripherals, clocks, and memory active and then switch to that mode.

[Chapter 6: Clock Description](#), includes a graphic of the clock architecture of the MCU. This can be used to determine how to configure the MC\_CGM module. In general software will configure the module to enable the required clocks and PLLs and route these to the active modules.

After these steps are complete it is possible to configure the input/output pins and the modules for the application.

### 1.7.4 Other features

The MC\_ME module manages low power modes and so it is likely that it will be used to switch into different configurations (module sets, clocks) depending on the application requirements.



The MCU includes two other features to improve the integrity of the application:

- It is possible to enable a software watchdog (SWT) immediately at reset or afterwards to help detect code runaway.
- Individual register settings can be protected from unintended writes using the features of the Register Protection module. The protected registers are shown in [Chapter 32: Register Protection](#).

Other integration functionality is provided by the System Status and Configuration Module (SSCM).

## 2 Introduction

### 2.1 The SPC560B54/6x microcontroller family

The SPC560B54/6x is a family of Power Architecture®-based microcontrollers that target automotive vehicle body applications such as:

- Central body electronics
- Vehicle body controllers
- Smart junction boxes
- Front modules
- Body peripherals
- Door control
- Seat control

The SPC560B54/6x family expands the range of the SPC560B microcontroller family. It provides the scalability needed to implement platform approaches and delivers the performance required by increasingly sophisticated software architectures. The advanced and cost-efficient host processor core of the SPC560B54/6x automotive controller family complies with the Power Architecture embedded category, and only implements the VLE (variable-length encoding) APU, providing improved code density. It operates at speeds of up to 64 MHz and offers high performance processing optimized for low power consumption. It also capitalizes on the available development infrastructure of current Power Architecture devices and is supported with software drivers, operating systems and configuration code to assist with users implementations.

This document describes the features of the family and options available within the family members, and highlights important electrical and physical characteristics of the device.

### 2.2 SPC560B54/6x device comparison

[Table 3](#) summarizes the SPC560B54/6x family of microcontrollers.

**Table 3. SPC560B54/6x family comparison<sup>(1)</sup>**

Feature	SPC560B54		SPC560B60		SPC560B64				
CPU	e200z0h								
Execution speed <sup>(2)</sup>	Up to 64 MHz								
Code flash memory	768 KB		1 MB		1.5 MB				
Data flash memory	64 (4 × 16) KB								
SRAM	64 KB		80 KB		96 KB				
MPU	8-entry								
eDMA	16 ch								
10-bit ADC	Yes								
dedicated <sup>(3)</sup>	7 ch	15 ch	7 ch	15 ch	29 ch	7 ch	15 ch	29 ch	29 ch
shared with 12-bit ADC	19 ch								

**Table 3. SPC560B54/6x family comparison<sup>(1)</sup> (continued)**

Feature	SPC560B54		SPC560B60			SPC560B64			
12-bit ADC	Yes								
dedicated <sup>(4)</sup>	5 ch								
shared with 10-bit ADC	19 ch								
Total timer I/O <sup>(5)</sup> eMIOS	37 ch, 16-bit	64 ch, 16-bit	37 ch, 16-bit	64 ch, 16-bit	64 ch, 16-bit	37 ch, 16-bit	64 ch, 16-bit	64 ch, 16-bit	64 ch, 16-bit
Counter / OPWM / ICOC <sup>(6)</sup>	10 ch								
O(I)PWM / OPWFMB / OPWMCB / ICOC <sup>(7)</sup>	7 ch								
O(I)PWM / ICOC <sup>(8)</sup>	7 ch	14 ch	7 ch	14 ch	14 ch	7 ch	14 ch	14 ch	14 ch
OPWM / ICOC <sup>(9)</sup>	13 ch	33 ch	13 ch	33 ch	33 ch	13 ch	33 ch	33 ch	33 ch
SCI (LINFlex)	4	8	4	8	10	4	8	10	10
SPI (DSPI)	3	5	3	5	6	3	5	6	6
CAN (FlexCAN)	6								
I2C	1								
32 KHz oscillator	Yes								
GPIO <sup>(10)</sup>	77	121	77	121	149	77	121	149	149
Debug	JTAG								N2+
Package	LQFP 100	LQFP 144	LQFP 100	LQFP 144	LQFP 176	LQFP 100	LQFP 144	LQFP 176	LBGA208 <sup>(11)</sup>

1. Feature set dependent on selected peripheral multiplexing; table shows example.
2. Based on 125 °C ambient operating temperature.
3. Not shared with 12-bit ADC, but possibly shared with other alternate functions.
4. Not shared with 10-bit ADC, but possibly shared with other alternate functions.
5. See the eMIOS section of the chip reference manual for information on the channel configuration and functions.
6. Each channel supports a range of modes including Modulus counters, PWM generation, Input Capture, Output Compare.
7. Each channel supports a range of modes including PWM generation with dead time, Input Capture, Output Compare.
8. Each channel supports a range of modes including PWM generation, Input Capture, Output Compare, Period and Pulse width measurement.
9. Each channel supports a range of modes including PWM generation, Input Capture, and Output Compare.
10. Maximum I/O count based on multiplexing with peripherals.
11. LBGA208 available only as development package for Nexus2+.

## 2.3 Device block diagram

*Figure 2* shows a top-level block diagram of the SPC560B54/6x.

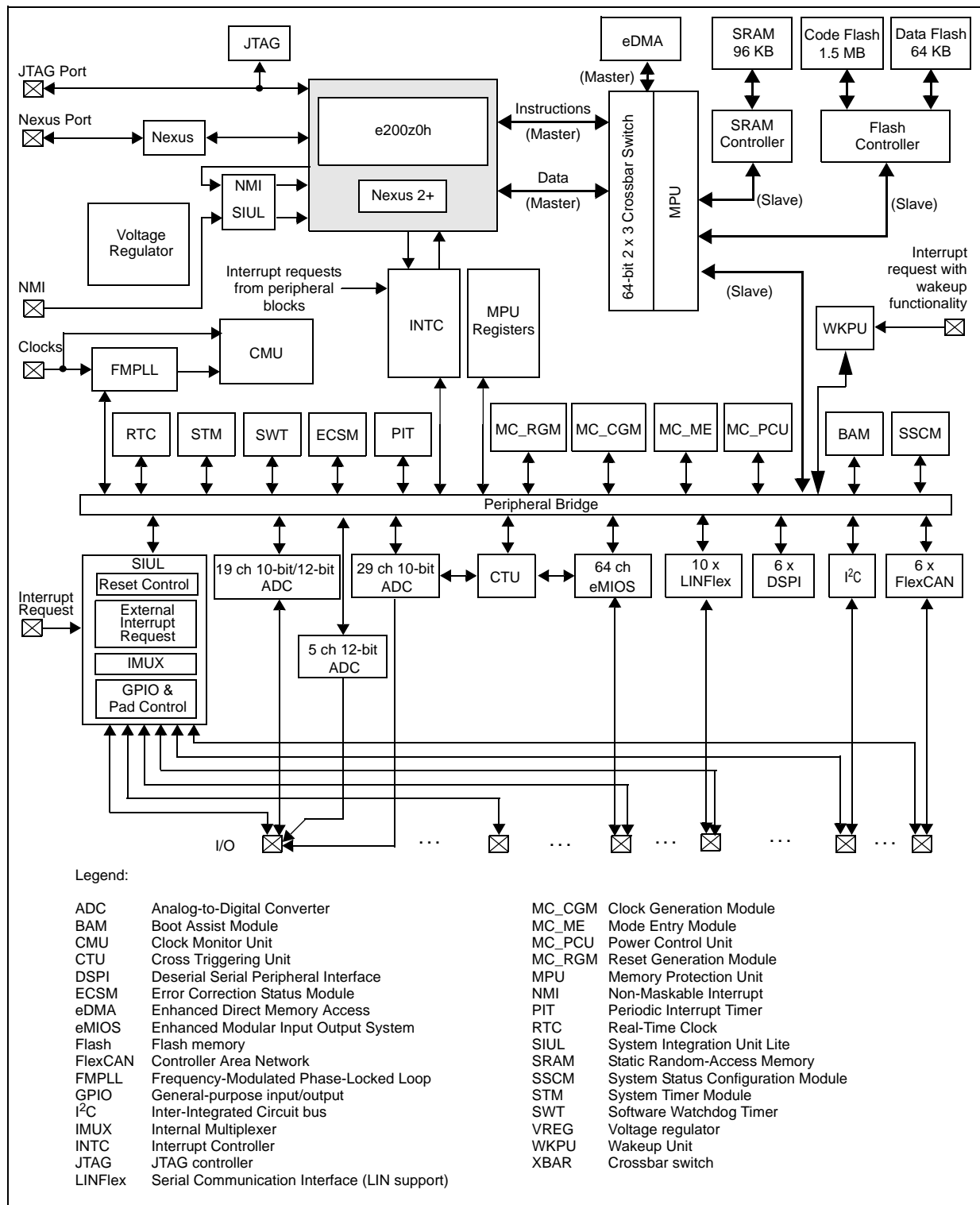


Figure 2. SPC560B54/6x block diagram

## 2.4 Feature details

### 2.4.1 e200z0h core processor

The e200z0h core includes the following features:

- High performance, low cost e200z0h core processor for managing peripherals and interrupts
- Single issue 4-stage pipelined in-order execution, 32-bit Power Architecture CPU
- Variable length encoding (VLE), allowing mixed 16-bit and 32-bit instructions
  - Results in efficient code size footprint
  - Minimizes impact on performance
- Branch processing acceleration using lookahead instruction buffer
- Load/store unit
  - 1-cycle load latency
  - Misaligned access support
  - No load-to-use pipeline bubbles
- 32-bit general purpose registers (GPRs)
- Separate instruction bus and load/store bus Harvard architecture
- Hardware vectored interrupt support
- Reservation instructions for implementing read-modify-write constructs
- Multi-cycle divide word (divw) and load multiple word (lmw) store multiple word (smw) multiple class instructions, can be interrupted to prevent increases in interrupt latency
- Extensive system development support through Nexus debug port

### 2.4.2 Crossbar switch (XBAR)

The following summarizes the SPC560B54/6x's implementation of the crossbar switch:

- 3 master ports:
  - CPU instruction bus
  - CPU load/store bus
  - eDMA
- Multiple bus slaves to enable access to flash memory, SRAM and peripherals
- Crossbar supports up to 2 consecutive transfers at any one time
- 32-bit internal address bus, 32-bit internal data bus
- Fixed priority arbitration based on port master

### 2.4.3 Interrupt Controller (INTC)

The SPC560B54/6x implements an interrupt controller that features the following:

- Unique 9-bit vector for each of the 231 separate interrupt sources
- 8 software triggerable interrupt sources
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Ability to modify the ISR or task priority
- External high priority interrupt directly accessing the main core critical interrupt mechanism

### 2.4.4 System Integration Unit Lite (SIUL)

The SIUL features the following:

- Up to 4 levels of internal pin multiplexing, allowing exceptional flexibility in the allocation of device functions for each package
- Centralized general purpose input output (GPIO) control of up to 149 input/output pins (package dependent)
- All GPIO pins independently configurable to support pull-up, pull down, or no pull
- Reading and writing to GPIO supported both as individual pins and 16-bit wide ports
- All peripheral pins can be alternatively configured as both general purpose input or output pins except ADC channels which support alternative configuration as general purpose inputs, with selected pins able to also support outputs
- Direct readback of the pin value supported on all digital output pins through the SIUL
- Configurable digital input filter that can be applied to up to 16 general purpose input pins for noise elimination on external interrupts
- Register configuration protected against change with soft lock for temporary guard or hard lock to prevent modification until next reset
- Support for two 32-bit virtual ports via the DSPI serialization

## 2.4.5 Flash memory

The on-chip flash memory on the SPC560B54/6x features the following:

- Up to 1.5 MB burst flash memory
  - 4 x 128-bit page buffers with programmable prefetch control
  - Typical flash-memory access time: 0 wait-state for buffer hits, 2 wait-states for page buffer miss at 64 MHz
  - Page buffers can be allocated for code-only, fixed partitions of code and data, all available for any access
  - 64-bit ECC with single-bit correction, double-bit detection for data integrity
- Censorship protection scheme to prevent flash-memory content visibility
- Separate dedicated Data Flash for EEPROM emulation
  - 4 erase sectors each containing 16 KB of memory
  - Offers read-while-write functionality from main program space
- Small block flash-memory arrangement in main array to support features such as boot block, operating system block
- Hardware managed flash memory writes, erase and verify sequence
- Error correction status
  - Configurable error-correcting codes (ECC) reporting for SRAM and flash memory
  - Supports optional reporting of single-bit errors
  - Protected mechanism for reporting of corrected ECC values
  - Error address recorded including Access type and Master
  - Flash-memory ECC reporting registers mirrored into ECSM address space but data comes from the flash-memory module
  - Flash-memory module can be interrogated to provide ECC bit error location
  - Margin read for flash-memory array supported for initial program verification

## 2.4.6 SRAM

The on-chip SRAM on the SPC560B54/6x features the following:

- Up to 96 KB general purpose SRAM
- Typical SRAM access time: 0 wait-state for reads and 32-bit writes; 1 wait-state for 8- and 16-bit writes if back to back with a read to same memory block
- 32-bit ECC with single-bit correction, double-bit detection for data integrity
- Supports byte (8-bit), half word (16-bit), and word (32-bit) writes for optimal use of memory
- User transparent ECC encoding and decoding for byte, half word, and word accesses
- Separate internal power domain applied to 32 KB SRAM block or 8 KB SRAM block during STANDBY modes to retain contents during low power mode

### 2.4.7 Memory Protection Unit (MPU)

The MPU provides the following features

- 8 region descriptors for per-master protection
- Start and end address defined with 32-byte granularity
- Overlapping regions supported
- Protection attributes can optionally include process ID
- Protection offered for 3 concurrent read ports
- Read and write attributes for all masters
- Execute and supervisor/user mode attributes for processor masters

### 2.4.8 Boot Assist Module (BAM)

The device implements a Boot Assist Module (BAM):

- Block of read-only memory containing VLE code which is executed according to boot mode of the device
- Download of code into internal SRAM possible via FlexCAN or LINFlex, after which code can be executed

### 2.4.9 Enhanced Modular Input Output System (eMIOS)

The SPC560B54/6x implements a scaled-down version of the eMIOS module:

- Up to 64 timed I/O channels with 16-bit counter resolution
- Buffered updates
- Support for shifted PWM outputs to minimize occurrence of concurrent edges
- Supports configurable trigger outputs for ADC conversion for synchronization to channel output waveforms
- Edge-aligned output pulse width modulation
  - Programmable pulse period and duty cycle
  - Supports 0% and 100% duty cycle
  - Shared or independent time bases
- DMA transfer support available

[Table 4](#) shows the supported eMIOS modes.

**Table 4. Supported eMIOS channel modes**

Mode		Channel type			
Description	Name	Counter / OPWM / ICOC	O(I)PWM / OPWFMB / OPWMCB / ICOC	O(I)PWM / ICOC	OPWM / ICOC
Double action output compare	DAOC	x	x	x	—
General purpose input / output	GPIO	x	x	x	x
Input filter	IPF	x	x	x	x
Input period measurement	IPM	x	x	x	—



**Table 4. Supported eMIOS channel modes (continued)**

Mode		Channel type			
Description	Name	Counter / OPWM / ICOC	O(I)PWM / OPWFMB / OPWMCB / ICOC	O(I)PWM / ICOC	OPWM / ICOC
Input pulse width measurement	IPWM	x	x	x	—
Modulus counter	MC	x	—	—	—
Modulus counter buffered (up / down)	MCB	x	x	—	—
Output pulse width and frequency modulation buffered	OPWFMB	x	x	—	—
Output pulse width modulation buffered	OPWMB	—	x	x	x
Center aligned output PWM buffered with dead time	OPWMCB	—	x	—	—
Output pulse width modulation trigger	OPWMT	x	x	x	x
Pulse edge accumulation	PEA	x	—	—	—
Pulse edge counting	PEC	x	—	—	—
Quadrature decode	QDEC	x	—	—	—
Single action input capture	SAIC	x	x	x	x
Single action output compare	SAOC	x	x	x	x

Table 5 shows the maximum eMIOS channel allocation.

**Table 5. eMIOS configuration**

Channel type	Maximum number of channels		Total
	eMIOS_0	eMIOS_1	
Counter / OPWM / ICOC <sup>(1)</sup>	5	5	10
O(I)PWM / OPWFMB / OPWMCB / ICOC <sup>(2)</sup>	7	0	7
O(I)PWM / ICOC <sup>(3)</sup>	7	7	14
OPWM / ICOC <sup>(4)</sup>	13	20	33

1. Each channel supports a range of modes including Modulus counters, PWM generation, Input Capture, Output Compare.
2. Each channel supports a range of modes including PWM generation with dead time, Input Capture, Output Compare.
3. Each channel supports a range of modes including PWM generation, Input Capture, Output Compare, Period and Pulse width measurement.
4. Each channel supports a range of modes including PWM generation, Input Capture, Output Compare.

## 2.4.10 Deserial Serial Peripheral Interface Module (DSPI)

The DSPI features the following:

- Up to 6 DSPI modules supported
- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- Programmable data frames from 4 to 16 bits
- Up to 6 chip select lines available, depending on package and pin multiplexing, to enable 64 external devices to be selected using external muxing from a single DSPI
- Up to 8 transfer types, independently configurable for each DSPI using the clock and transfer attributes registers
- Chip select strobe available as alternate function on one of the chip select pins for deglitching
- FIFOs for buffering up to 4 transfers on the transmit and receive side
- General purpose I/O functionality on pins when not used for SPI
- Queueing operation possible through use of eDMA
- 32-bit serialization of data enabling virtual GPIO ports on 2 DSPI modules

### 2.4.11 Controller Area Network module (FlexCAN)

The enhanced FlexCAN module features the following:

- Up to 6 FlexCAN modules supported
- Compliant with CAN protocol specification, version 2.0B active
- 64 mailboxes per FlexCAN module
  - Mailboxes configurable while module remains synchronized to CAN bus
  - Each mailbox configurable as transmit or receive
- Transmit features
  - Supports configuration of multiple mailboxes to form message queues of scalable depth
  - Arbitration scheme according to message ID or message buffer number
  - Internal arbitration to guarantee no inner or outer priority inversion
  - Transmit abort procedure and notification
- Receive features
  - Individual programmable filters for each mailbox
  - 8 mailboxes configurable as a 6-entry receive FIFO
  - 8 programmable acceptance filters for receive FIFO
- Programmable clock source
  - System clock
  - Direct oscillator clock to avoid PLL jitter
- Listen-only mode capabilities
- CAN sampler available for connection to one of available CAN module pads
  - Supports capturing of first message identifier while in STOP or STANDBY modes

### 2.4.12 System clocks and clock generation

The following list summarizes the system clock and clock generation on the SPC560B54/6x:

- System clock can be derived from the following sources
  - External crystal oscillator
  - FMPLL
  - 16 MHz fast internal RC oscillator
- Programmable output clock divider of system clock ( $\div 1$ ,  $\div 2$ ,  $\div 4$ )
- Separate programmable peripheral bus clock divider ratio ( $\div 1$ ,  $\div 2$ ,  $\div 4$ ) applied to system clock
- Frequency modulated phase-locked loop (FMPLL)
  - Input clock frequency from 4 MHz to 16 MHz
  - Clock source: external oscillator
  - Lock detect circuitry continuously monitors lock status
  - Loss of clock (LOC) detection for reference and feedback clocks
  - On-chip loop filter
    - Improves electromagnetic interference performance
    - Reduces number of external components required
- On-chip fast external crystal oscillator supporting 4 MHz to 16 MHz
- Dedicated 16 MHz fast internal RC oscillator
  - Used as default clock source out of reset
  - Provides clock for rapid startup from low power modes
  - Provides back-up clock in the event of FMPLL or external oscillator clock failure
  - Offers independent clock source for the watchdog timer
  - 5% accuracy over the operating temperature range
  - Trimming registers to support frequency adjustment with in-application calibration
- Dedicated 128 kHz slow internal RC oscillator for low power mode operation and self wakeup
  - 10% accuracy
  - Trimming registers to support improve accuracy with in-application calibration
- 32 KHz slow external crystal oscillator for low power real-time clock

### 2.4.13 System timers

The system timers include:

- Periodic Interrupt Timer (PIT) timers (including ADC trigger)
- 1 Real-Time Counter (RTC) timer

The PIT is an array of timers that can be used to raise interrupts, trigger CTU channels, and ADC conversions. The RTC supports wakeup from low power modes or real-time clock generation.

### 2.4.13.1 Periodic Interrupt Timer module (PIT)

The PIT features the following:

- 8 general purpose interrupt timers
- Up to 2 interrupt timers for triggering ADC injected conversions (one for 10-bit ADC, one for 12-bit ADC)
- Up to 4 interrupt timers for triggering DMA transfers
- Up to 2 interrupt timers for triggering CTU
- 32-bit counter resolution
- Clocked by system clock frequency

### 2.4.13.2 Real-Time Counter (RTC)

The RTC features the following:

- Configurable resolution for different timeout periods
  - 1 sec resolution for > 1 hour period
  - 1 ms resolution for 2 second period
- Selectable clock sources
  - 32 KHz slow external crystal oscillator
  - 128 kHz slow internal RC oscillator
  - Divided 16 MHz fast internal RC oscillator
- Supports continued operation through all resets except POR (power-on reset)

### 2.4.14 System watchdog timer

The watchdog on the SPC560B54/6x features the following:

- Activation by software or out of reset
- 32-bit modulus counter
- Clock source: robust 128 kHz slow internal RC oscillator (divisible by 1 to 32)
- Supports normal or windowed mode
- Configurable response on timeout: reset, interrupt, or interrupt followed by reset
- Reset by writing a software key to memory mapped register
- Support for protected access to watchdog control registers with optional soft and hard locks
  - Soft lock allows temporary locking of configuration
  - Once enabled, hard lock prevents any changes until after a reset
- Supports halting during low power modes

### 2.4.15 Inter-Integrated Circuit (I<sup>2</sup>C) module

The I<sup>2</sup>C module features the following:

- 1 I<sup>2</sup>C module supported
- 2-wire bidirectional serial bus for on-board communications
- Compatibility with I<sup>2</sup>C bus standard
- Multimaster operation
- Software-programmable for one of 256 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

### 2.4.16 On-chip voltage regulator (VREG)

The on-chip voltage regulator includes the following features:

- Optional support for internal and external ballast resistor based on power consumption
- Regulates 3.3 or 5 V  $\pm 10\%$  input to generate all internal supplies for internal control
- Manages power gating
- Low power regulators support operation when in STOP and STANDBY modes to minimize power consumption
- Fast startup on-chip regulators for rapid exit from low power modes
- Low voltage reset supported on all internal supplies

### 2.4.17 Analog-to-Digital Converter (ADC)

The ADC features the following:

- 2 ADC modules, one 10-bit resolution and one 12-bit resolution supporting synchronous conversions on channels
- $0-V_{DD}$  common mode conversion range
- Independent reference supplies for each ADC
- Conversions times of  $< 1 \mu s$  available
- Up to 53 single ended inputs channels, expandable to 81 channels with external multiplexers
- Up to 19 shared channels, among which, 16 called ANP are mapped on dedicated pins, not multiplexed with any other functionality, in order to improve the accuracy. All other channels, called ANS or ANX are multiplexed with other functionalities.
  - Up to 19 channels shared between 10-bit and 12-bit ADCs
  - Up to 5 dedicated 12-bit ADC channels
  - Up to 29 dedicated 10-bit ADC channels
- Externally multiplexed channels
  - Internal control to support generation of external analog multiplexor selection
  - 4 internal channels optionally used to support externally multiplex inputs, providing transparent control for additional ADC channels
  - Each of the 3 channels supports up to 8 externally muxed inputs
  - Individual dedicated result register also available for externally muxed conversion channels
  - 3 independently configurable sample and conversion times for high occurrence channels, internally muxed channels and externally muxed channels
- Configurable right-aligned or left-aligned result formats
- Support for one-shot, scan and injection conversion modes
- Independently configurable parameters for channels:
  - Offset refresh
  - Sampling
- Conversion triggering support
  - Internal conversion triggering from periodic interrupt timer (PIT) or timed I/O module (eMIOS) through cross triggering unit (CTU)
  - Internal conversion triggering from periodic interrupt timer (PIT)
  - 1 input pin configurable as external conversion trigger source
- Up to 6 configurable analog comparator channels offering range comparison with triggered alarm
  - Greater than
  - Less than
  - Out of range
- All unused analog pins available as general purpose input pins
- Unused 10-bit ADC analog pins, with the exception of the 19 dedicated high accuracy channels, available as general purpose output pins
- Power-down mode

- Supports DMA transfer of results based on end of conversion chain or each conversion
- Separate dedicated DMA request for injection mode

### 2.4.18 Enhanced Direct Memory Access controller (eDMA)

The following summarizes the SPC560B54/6x's implementation of the eDMA controller:

- 16 channels to support independent 8-, 16-, or 32-bit single value or block transfers
- Support of variable sized queues and circular queues
- Source and destination address registers independently configured to post-increment or remain constant
- Each transfer initiated by peripheral, CPU, periodic timer interrupt or eDMA channel request
- Peripheral DMA request sources possible from SPIs, I<sup>2</sup>C, 10-bit ADC, 12-bit ADC, eMIOS and GPIOs
- Each eDMA channel able to optionally send interrupt request to CPU on completion of single value or block transfer
- DMA transfers possible between system memories and all accessible memory mapped locations including peripheral and registers
- Programmable DMA Channel Mux allows assignment of any DMA source to any available DMA channel with total of up to 64 potential request sources

### 2.4.19 Cross Trigger Unit (CTU)

The CTU enables the synchronization of ADC conversions with a timer event. Its key features are:

- Single cycle delayed trigger output triggered by up to 64 input flags/events connected to different timers in the system
- Triggers ADC conversions from any eMIOS channel
- Triggers ADC conversions from up to 2 dedicated PITs
- Maskable interrupt generation whenever a trigger output is generated
- 1 event configuration register dedicated to each timer event allows to define the corresponding ADC channel
- Acknowledgment signal to eMIOS/PIT for clearing the flag
- Synchronization with ADC to avoid collision



## 2.4.20 Serial communication interface module (LINFlex)

The LINFlex on the SPC560B54/6x features the following:

- Up to 10 LINFlex modules supported
- Supports LIN master mode, LIN slave mode and UART mode
- DMA connected on LINFlex\_0 and LINFlex\_1
- LINFlex\_0 supporting LIN master and slave mode; LINFlex\_1 to LINFlex\_9 modules supporting LIN master mode
- LIN state machine compliant to LIN 1.3, 2.0 and 2.1 specifications
- Handles LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store identified and up to 8 data bytes
  - Supports message length of up to 64 bytes
  - Detection and flagging of LIN errors
    - Sync field
    - Delimiter
    - ID parity
    - Bit, framing
    - Checksum and timeout errors
  - Classic or extended checksum calculation
  - Configurable break duration of up to 36-bit times
  - Programmable baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features
    - Loop back
    - Self Test
    - LIN bus stuck dominant detection
  - Interrupt driven operation with 16 interrupt sources
- LIN slave mode features
  - Autonomous LIN header handling
  - Autonomous LIN response handling
  - 16 identifiers filters for discarding irrelevant LIN frames
- UART mode
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-bytes receive, 4-bytes transmit
  - Configurable word length (8-bit or 9-bit words)
  - Error detection and flagging
    - Parity, noise and framing errors
  - Interrupt driven operation with 4 interrupt sources
  - Separate transmitter and receiver CPU interrupt sources
  - 16-bit programmable baud rate modulus counter and 16-bit fractional
  - 2 receiver wakeup methods

SPC560B54/6x devices include two functionally different LINFlex controller types. These are distinguished in the documentation by the abbreviations “LINFlex” and “LINFlexD”. The latter name represents the DMA support available on this controller type. The SPC560B54/6x devices combine these two types to provide up to 10 modules supporting the LINFlex protocol. [Table 6](#) shows the module (instance) numbers and the corresponding functional controller type.

**Table 6. LINFlex numbering and naming**

Module numbers	Module version
0 and 1	LINFlexD
2–9	LINFlex

### 2.4.21 Nexus Port Controller (NPC)

Nexus features the following:

- According to IEEE-ISTO 5001-2003
- Nexus 2+ features supported
  - Static debug
  - Watchpoint messaging
  - Ownership trace messaging
  - Program trace messaging
  - Real-time read/write of any internally memory mapped resources through JTAG pins
  - Overrun control, which selects whether to stall before Nexus overruns or keep executing and allow overwrite of information
  - Watchpoint triggering, watchpoint triggers program tracing
- Configured via the IEEE 1149.1 (JTAG) port
- Nexus auxiliary port supported on LPGA208 packages

### 2.4.22 JTAG Controller (JTAGC)

JTAG features the following:

- JTAG low pin count interface (IEEE 1149.1) test access port (TAP) interface
- Backward compatible to standard JTAG IEEE 1149.1-2001 test access port (TAP) interface
- Supports boundary scan testing
- All JTAG pins reusable in application as standard IOs

## 2.5 Developer support

The SPC560B54/6x MCU family uses tools and third-party developers which offer a widespread, established network of tool and software vendors. It also features a high-performance Nexus debug interface.

The following development support will be available:

- Automotive evaluation boards (EVB) featuring CAN, LIN interfaces, and more
- Compilers
- Debuggers
- JTAG and Nexus interfaces

The following software support will be available:

- OSEK solutions will be available from multiple third parties
- CAN and LIN drivers
- AutoSAR package

### 3 Memory Map

[Table 7](#) shows the memory map for the SPC560B54/6x. All addresses on the device, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each IP block.

**Table 7. SPC560B54/6x memory map**

Start address	End address	Size (KB)	Region name
0x0000_0000	0x0000_7FFF	32	Code flash memory array 0
0x0000_8000	0x0000_BFFF	16	Code flash memory array 0
0x0000_C000	0x0000_FFFF	16	Code flash memory array 0
0x0001_0000	0x0001_7FFF	32	Code flash memory array 0
0x0001_8000	0x0001_FFFF	32	Code flash memory array 0
0x0002_0000	0x0003_FFFF	128	Code flash memory array 0
0x0004_0000	0x0005_FFFF	128	Code flash memory array 0
0x0006_0000	0x0007_FFFF	128	Code flash memory array 0
0x0008_0000	0x0009_FFFF	128	Code flash memory array 1
0x000A_0000	0x000B_FFFF	128	Code flash memory array 1
0x000C_0000	0x000D_FFFF	128	Code flash memory array 1
0x000E_0000	0x000F_FFFF	128	Code flash memory array 1
0x0010_0000	0x0011_FFFF	128	Code flash memory array 2
0x0012_0000	0x0013_FFFF	128	Code flash memory array 2
0x0014_0000	0x0015_FFFF	128	Code flash memory array 2
0x0016_0000	0x0017_FFFF	128	Code flash memory array 2
0x0018_0000	0x001F_FFFF	512	Reserved
0x0020_0000	0x0020_3FFF	16	Flash memory shadow array
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code flash memory array 0 test sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_3FFF	16	Data flash memory array 0
0x0080_4000	0x0080_7FFF	16	Data flash memory array 0
0x0080_8000	0x0080_BFFF	16	Data flash memory array 0
0x0080_C000	0x0080_FFFF	16	Data flash memory array 0
0x0081__0000	0x00BF_FFFF	4032	Reserved
0x00C0_0000	0x00C0_3FFF	16	Data flash memory array 0 test sector
0x00C0_4000	0x00FF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Flash memory emulation mapping
0x2000_0000	0x3FFF_FFFF	524288	Reserved

**Table 7. SPC560B54/6x memory map (continued)**

Start address	End address	Size (KB)	Region name
0x4000_0000	0x4001_7FFF	96	SRAM
0x4001_8000	0xBFFF_FFFF	2097120	Reserved
Off-platform peripherals			
0xC000_0000	0xC3F8_7FFF	65056	Reserved
0xC3F8_8000	0xC3F8_BFFF	16	Code flash memory A configuration
0xC3F8_C000	0xC3F8_FFFF	16	Data flash memory A configuration
0xC3F9_0000	0xC3F9_3FFF	16	SIUL
0xC3F9_4000	0xC3F9_7FFF	16	WKPU
0xC3F9_8000	0xC3F9_FFFF	32	Reserved
0xC3FA_0000	0xC3FA_3FFF	16	eMIOS_0
0xC3FA_4000	0xC3FA_7FFF	16	eMIOS_1
0xC3FA_8000	0xC3FD_7FFF	192	Reserved
0xC3FD_8000	0xC3FD_BFFF	16	SSCM
0xC3FD_C000	0xC3FD_FFFF	16	MC_ME
0xC3FE_0000	0xC3FE_3FFF	16	MC_CGM
0xC3FE_4000	0xC3FE_7FFF	16	MC_RGM
0xC3FE_8000	0xC3FE_BFFF	16	MC_PCU
0xC3FE_C000	0xC3FE_FFFF	16	RTC/API
0xC3FF_0000	0xC3FF_3FFF	16	PIT
0xC3FF_4000	0xFFDF_FFFF	981040	Reserved
0xFFE0_0000	0xFFE0_3FFF	16	ADC_0
0xFFE0_4000	0xFFE0_7FFF	16	ADC_1
0xFFE0_8000	0xFFE2_FFFF	160	Reserved
0xFFE3_0000	0xFFE3_3FFF	16	I2C_0
0xFFE3_4000	0xFFE3_FFFF	48	Reserved
0xFFE4_0000	0xFFE4_3FFF	16	LINFlex_0
0xFFE4_4000	0xFFE4_7FFF	16	LINFlex_1
0xFFE4_8000	0xFFE4_BFFF	16	LINFlex_2
0xFFE4_C000	0xFFE4_FFFF	16	LINFlex_3
0xFFE5_0000	0xFFE5_3FFF	16	LINFlex_4
0xFFE5_4000	0xFFE5_7FFF	16	LINFlex_5
0xFFE5_8000	0xFFE5_BFFF	16	LINFlex_6
0xFFE5_C000	0xFFE5_FFFF	16	LINFlex_7
0xFFE6_0000	0xFFE6_3FFF	16	Reserved
0xFFE6_4000	0xFFE6_7FFF	16	CTU

Table 7. SPC560B54/6x memory map (continued)

Start address	End address	Size (KB)	Region name
0xFFE6_8000	0xFFE6_FFFF	32	Reserved
0xFFE7_0000	0xFFE7_3FFF	16	CAN sampler
0xFFE7_4000	0xFFE7_FFFF	48	Reserved
0xFFE8_0000	0xFFEF_FFFF	512	Mirrored range 0x3F80000—0xC3FFFFFF
0xFFFF0_0000	0xFFFF0_FFFF	64	Reserved
0xFFFF1_0000	0xFFFF1_3FFF	16	MPU
0xFFFF1_4000	0xFFFF3_7FFF	144	Reserved
0xFFFF3_8000	0xFFFF3_BFFF	16	SWT
0xFFFF3_C000	0xFFFF3_FFFF	16	STM
0xFFFF4_0000	0xFFFF4_3FFF	16	ECSM
0xFFFF4_4000	0xFFFF4_7FFF	16	eDMA
0xFFFF4_8000	0xFFFF4_BFFF	16	INTC
0xFFFF4_C000	0xFFFF8_FFFF	272	Reserved
0xFFFF9_0000	0xFFFF9_3FFF	16	DSPI_0
0xFFFF9_4000	0xFFFF9_7FFF	16	DSPI_1
0xFFFF9_8000	0xFFFF9_BFFF	16	DSPI_2
0xFFFF9_C000	0xFFFF9_FFFF	16	DSPI_3
0xFFFFA_0000	0xFFFFA_3FFF	16	DSPI_4
0xFFFFA_4000	0xFFFFA_7FFF	16	DSPI_5
0xFFFFA_8000	0xFFFFA_FFFF	32	Reserved
0xFFFFB_0000	0xFFFFB_3FFF	16	LINFlex_8
0xFFFFB_4000	0xFFFFB_7FFF	16	LINFlex_9
0xFFFFB_8000	0xFFFFB_FFFF	32	Reserved
0xFFFFC_0000	0xFFFFC_3FFF	16	FlexCAN_0
0xFFFFC_4000	0xFFFFC_7FFF	16	FlexCAN_1
0xFFFFC_8000	0xFFFFC_BFFF	16	FlexCAN_2
0xFFFFC_C000	0xFFFFC_FFFF	16	FlexCAN_3
0xFFFFD_0000	0xFFFFD_3FFF	16	FlexCAN_4
0xFFFFD_4000	0xFFFFD_7FFF	16	FlexCAN_5
0xFFFFD_8000	0xFFFFD_BFFF	16	Reserved
0xFFFFD_C000	0xFFFFD_FFFF	16	DMA_MUX
0xFFFFE_0000	0xFFFFF_BFFF	144	Reserved
0xFFFFF_C000	0xFFFFF_FFFF	16	BAM

# 4 Signal description

## 4.1 Package pinouts

Figure 3, Figure 4, Figure 5, and Figure 6 show the location of the signals on the packages that this chip is available in.

For more information on pin multiplexing on this chip, see Table 8 through Table 11.

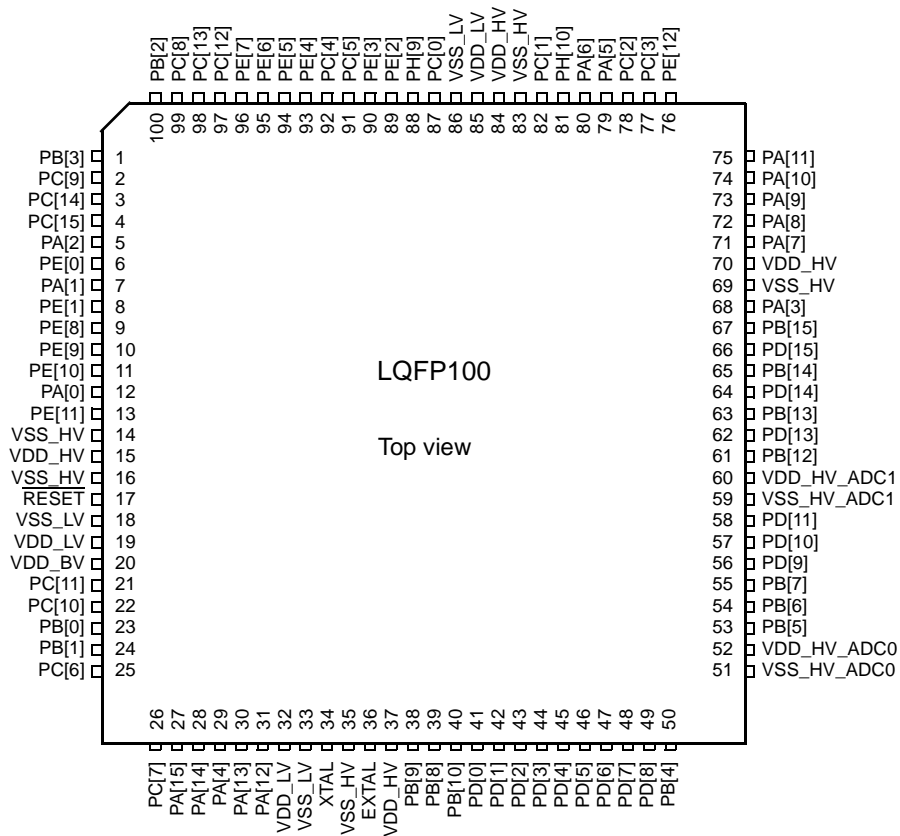


Figure 3. 100-pin LQFP pinout

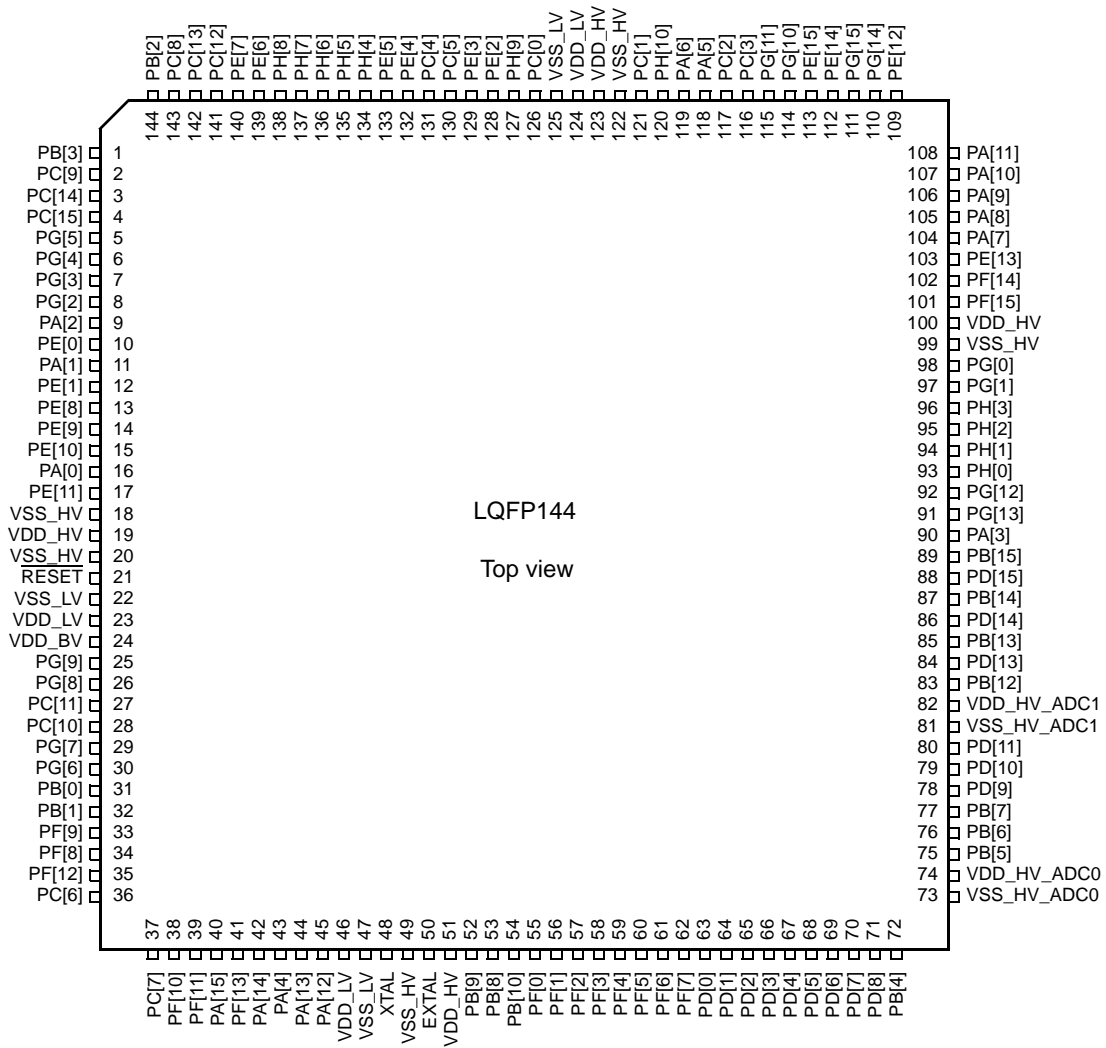


Figure 4. 144-pin LQFP pinout



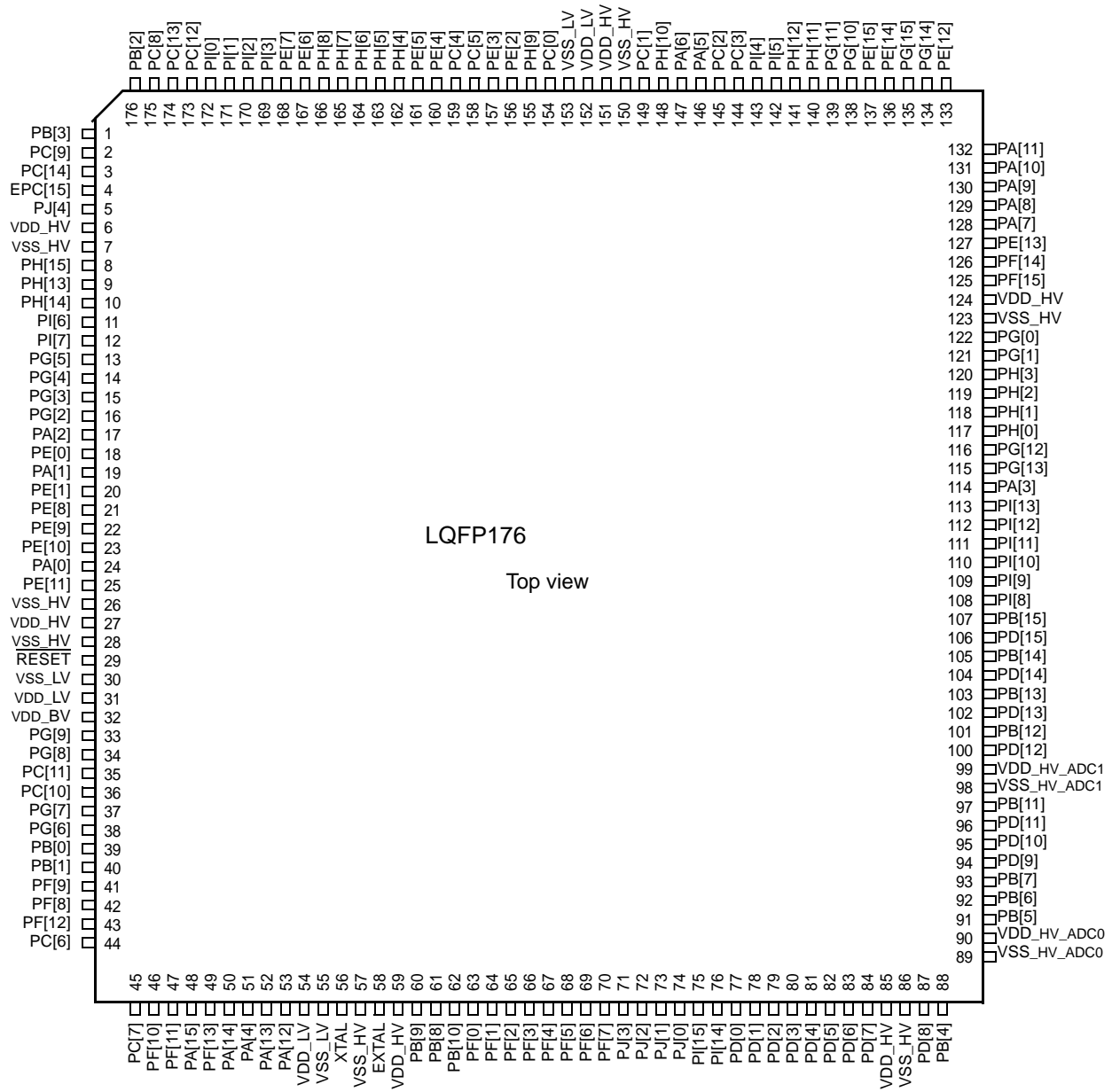


Figure 5. 176-pin LQFP pinout

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16				
A	PC[8]	PC[13]	PH[15]	PJ[4]	PH[8]	PH[4]	PC[5]	PC[0]	PI[0]	PI[1]	PC[2]	PI[4]	PE[15]	PH[11]	NC	NC	A			
B	PC[9]	PB[2]	PH[13]	PC[12]	PE[6]	PH[5]	PC[4]	PH[9]	PH[10]	PI[2]	PC[3]	PG[11]	PG[15]	PG[14]	PA[11]	PA[10]	B			
C	PC[14]	VDD_H V	PB[3]	PE[7]	PH[7]	PE[5]	PE[3]	VSS_LV	PC[1]	PI[3]	PA[5]	PI[5]	PE[14]	PE[12]	PA[9]	PA[8]	C			
D	PH[14]	PI[6]	PC[15]	PI[7]	PH[6]	PE[4]	PE[2]	VDD_L V	VDD_H V	NC	PA[6]	PH[12]	PG[10]	PF[14]	PE[13]	PA[7]	D			
E	PG[4]	PG[5]	PG[3]	PG[2]									PG[1]	PG[0]	PF[15]	VDD_H V	E			
F	PE[0]	PA[2]	PA[1]	PE[1]									PH[0]	PH[1]	PH[3]	PH[2]	F			
G	PE[9]	PE[8]	PE[10]	PA[0]	VSS_H V				VSS_H V	VSS_H V	VSS_H V	VSS_H V	VDD_H V				PI[12]	PI[13]	MSEO	G
H	VSS_HV	PE[11]	VDD_H V	NC	VSS_H V				VSS_H V	VSS_H V	VSS_H V	VSS_H V	MDO3				MDO2	MDO0	MDO1	H
J	RESET	VSS_LV	NC	NC	VSS_H V				VSS_H V	VSS_H V	VSS_H V	VSS_H V	PI[8]				PI[9]	PI[10]	PI[11]	J
K	EVTI	NC	VDD_B V	VDD_L V	VSS_H V				VSS_H V	VSS_H V	VSS_H V	VSS_H V	VDD_H V_ADC 1				PG[12]	PA[3]	PG[13]	K
L	PG[9]	PG[8]	NC	EVTO									PB[15]				PD[15]	PD[14]	PB[14]	L
M	PG[7]	PG[6]	PC[10]	PC[11]									PB[13]				PD[13]	PD[12]	PB[12]	M
N	PB[1]	PF[9]	PB[0]	VDD_H V	PJ[0]	PA[4]	VSS_LV	EXTAL	VDD_H V	PF[0]	PF[4]	VSS_H V_ADC 1	PB[11]				PD[10]	PD[9]	PD[11]	N
P	PF[8]	PJ[3]	PC[7]	PJ[2]	PJ[1]	PA[14]	VDD_L V	XTAL	PB[10]	PF[1]	PF[5]	PD[0]	PD[3]	VDD_H V_ADC 0	PB[6]				PB[7]	P
R	PF[12]	PC[6]	PF[10]	PF[11]	VDD_H V	PA[15]	PA[13]	PI[14]	XTAL32	PF[3]	PF[7]	PD[2]	PD[4]	PD[7]	VSS_H V_ADC 0	PB[5]				R
T	NC	NC	NC	MCKO	NC	PF[13]	PA[12]	PI[15]	EXTAL 32	PF[2]	PF[6]	PD[1]	PD[5]	PD[6]	PD[8]	PB[4]				T

NOTE: The LBG208 is available only as development package for Nexus 2+.

NC = Not connected

Figure 6. 208-pin BGA pinout

### 4.2 Pad configuration during reset phases

All pads have a fixed configuration under reset.

During the power-up phase, all pads are forced to tristate.

After power-up phase, all pads are tristate with the following exceptions:

- PA[9] (FAB) is pull-down. Without external strong pull-up the device starts fetching from flash.
- PA[8], PC[0] and PH[9:10] are in input weak pull-up when out of reset.
- RESET pad is driven low by the device till 40 FIRC clock cycles after phase2 completion. Minimum phase3 duration is 40 FIRC cycles.
- Nexus output pads (MDO[n], MCKO, EVTO, MSEO) are forced to output.

### 4.3 Pad configuration during standby mode exit

Pad configuration (input buffer enable, pull enable) for low power wakeup pads are controlled by both SIUL and WKPU modules. During standby exit, all low power pads PA[0,1,2,4,15], PB[1,3,8,9,10]<sup>(a)</sup>, PC[7,9,11], PD[0,1], PE[0,9,11], PF[9,11,13]<sup>(b)</sup>, PG[3,5,7,9]<sup>(b)</sup>, PI[1,3]<sup>(c)</sup> are configured according to their respective configuration done in WKPU module. All other pads will have the same configuration as expected after a reset.

The TDO pad has been moved into the STANDBY domain in order to allow low-power debug handshaking in STANDBY mode. However, no pull-resistor is active on the TDO pad while in STANDBY mode. At this time the pad is configured as an input. When no debugger is connected the TDO pad is floating causing additional current consumption.

To avoid the extra consumption TDO must be connected. An external pull-up resistor in the range of 47-100 kOhms should be added between the TDO pin and VDD. Only in case the TDO pin is used as application pin and a pull-up cannot be used then a pull-down resistor with the same value should be used between TDO pin and GND instead.

### 4.4 Voltage supply pins

Voltage supply pins are used to provide power to the device. Three dedicated pins are used for 1.2 V regulator stabilization.

**Table 8. Voltage supply pin descriptions**

Port pin	Function	Pin number			
		LQFP100	LQFP144	LQFP176	LBGA208
VDD_HV	Digital supply voltage	15, 37, 70, 84	19, 51, 100, 123	6, 27, 59, 85, 124, 151	C2, D9, E16, G13, H3, N4, N9, R5
VSS_HV	Digital ground	14, 16, 35, 69, 83	18, 20, 49, 99, 122	7, 26, 28, 57, 86, 123, 150	G7, G8, G9, G10, H7, H8, H9, H10, J7, J8, J9, J10, K7, K8, K9, K10

- a. PB[8, 9] ports have wakeup functionality in all modes except STANDBY.
- b. PF[9,11,13],PG[3,5,7,9], PI[1,3] are not available in the 100 pin package.
- c. PI [1,3] are not available in the 144 pin package.

Table 8. Voltage supply pin descriptions (continued)

Port pin	Function	Pin number			
		LQFP100	LQFP144	LQFP176	LBGA208
VDD_LV	1.2 V decoupling pins. Decoupling capacitor must be connected between these pins and the nearest VSS_LV pin. <sup>(1)</sup>	19, 32, 85	23, 46, 124	31, 54, 152	D8, K4, P7
VSS_LV	1.2 V decoupling pins. Decoupling capacitor must be connected between these pins and the nearest VDD_LV pin. <sup>(1)</sup>	18, 33, 86	22, 47, 125	30, 55, 153	C8, J2, N7
VDD_BV	Internal regulator supply voltage	20	24	32	K3
VSS_HV_ADC0	Reference ground and analog ground for the A/D converter 0 (10 bit)	51	73	89	R15
VDD_HV_ADC0	Reference voltage and analog supply for the A/D converter 0 (10 bit)	52	74	90	P14
VSS_HV_ADC1	Reference ground and analog ground for the A/D converter 1 (12 bit)	59	81	98	N12
VDD_HV_ADC1	Reference voltage and analog supply for the A/D converter 1 (12-bit)	60	82	99	K13

1. A decoupling capacitor must be placed between each of the three VDD\_LV/VSS\_LV supply pairs to ensure stable voltage (see the recommended operating conditions in the device datasheet).

## 4.5 Pad types

In the device the following types of pads are available for system pins and functional port pins:

- S = Slow<sup>(d)</sup>
- M = Medium<sup>(d) (e)</sup>
- F = Fast<sup>(d) (e)</sup>
- I = Input only with analog feature<sup>(d)</sup>
- J = Input/Output with analog feature
- X = Oscillator

d. See the I/O pad electrical characteristics in the chip datasheet for details.

e. All medium and fast pads are in slow configuration by default at reset and can be configured as fast or medium. The only exception is PC[1] which is in medium configuration by default (see the PCR.SRC description in the chip reference manual, Pad Configuration Registers (PCR0—PCR148)).

## 4.6 System pins

The system pins are listed in [Table 9](#).

**Table 9. System pin descriptions**

Port pin	Function	I/O direction	Pad type	RESET config.	Pin number			
					LQFP 100	LQFP 144	LQFP 176	LBGA 208 <sup>(1)</sup>
RESET	Bidirectional reset with Schmitt-Trigger characteristics and noise filter.	I/O	M	Input weak pull-up after RGM PHASE2 and 40 FIRC cycles	17	21	29	J1
EXTAL	Analog output of the oscillator amplifier circuit, when the oscillator is not in bypass mode. Analog input for the clock generator when the oscillator is in bypass mode.	I/O	X	Tristate	36	50	58	N8
XTAL	Analog input of the oscillator amplifier circuit. Needs to be grounded if oscillator bypass mode is used.	I	X	Tristate	34	48	56	P8

1. LBGA208 available only as development package for Nexus2+

## 4.7 Functional port pins

The functional port pins are listed in [Table 10](#).

**Table 10. Functional port pin descriptions**

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction <sup>(2)</sup>	Pad type	RESET config. <sup>(3)</sup>	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208 <sup>(4)</sup>
Port A											
PA[0]	PCR[0]	AF0 AF1 AF2 AF3 —	GPIO[0] E0UC[0] CLKOUT E0UC[13] WKPU[19] <sup>(5)</sup>	SIUL eMIOS_0 MC_CGM eMIOS_0 WKPU	I/O I/O O I/O I	M	Tristate	12	16	24	G4

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PA[1]	PCR[1]	AF0 AF1 AF2 AF3 —	GPIO[1] E0UC[1] NMI(6) — WKPU[2](5)	SIUL eMIOS_0 WKPU — WKPU	I/O I/O I — I	S	Tristate	7	11	19	F3
PA[2]	PCR[2]	AF0 AF1 AF2 AF3 —	GPIO[2] E0UC[2] — MA[2] WKPU[3](5)	SIUL eMIOS_0 — ADC_0 WKPU	I/O I/O — O I	S	Tristate	5	9	17	F2
PA[3]	PCR[3]	AF0 AF1 AF2 AF3 — —	GPIO[3] E0UC[3] LIN5TX CS4_1 EIRQ[0] ADC1_S[0]	SIUL eMIOS_0 LINFlex_5 DSPI_1 SIUL ADC_1	I/O I/O O O I I	J	Tristate	68	90	114	K15
PA[4]	PCR[4]	AF0 AF1 AF2 AF3 — —	GPIO[4] E0UC[4] — CS0_1 LIN5RX WKPU[9](5)	SIUL eMIOS_0 — DSPI_1 LINFlex_5 WKPU	I/O I/O — I/O I I	S	Tristate	29	43	51	N6
PA[5]	PCR[5]	AF0 AF1 AF2 AF3	GPIO[5] E0UC[5] LIN4TX —	SIUL eMIOS_0 LINFlex_4 —	I/O I/O O —	M	Tristate	79	118	146	C11
PA[6]	PCR[6]	AF0 AF1 AF2 AF3 — —	GPIO[6] E0UC[6] — CS1_1 EIRQ[1] LIN4RX	SIUL eMIOS_0 — DSPI_1 SIUL LINFlex_4	I/O I/O — O I I	S	Tristate	80	119	147	D11

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PA[7]	PCR[7]	AF0 AF1 AF2 AF3 — —	GPIO[7] E0UC[7] LIN3TX — EIRQ[2] ADC1_S[1]	SIUL eMIOS_0 LINFlex_3 — SIUL ADC_1	I/O I/O O — I I	J	Tristate	71	104	128	D16
PA[8]	PCR[8]	AF0 AF1 AF2 AF3 — N/A(7) —	GPIO[8] E0UC[8] E0UC[14] — EIRQ[3] ABS[0] LIN3RX	SIUL eMIOS_0 eMIOS_0 — SIUL BAM LINFlex_3	I/O I/O I/O — I I I	S	Input, weak pull-up	72	105	129	C16
PA[9]	PCR[9]	AF0 AF1 AF2 AF3 N/A(7)	GPIO[9] E0UC[9] — CS2_1 FAB	SIUL eMIOS_0 — DSPI_1 BAM	I/O I/O — O I	S	Pull-down	73	106	130	C15
PA[10]	PCR[10]	AF0 AF1 AF2 AF3 —	GPIO[10] E0UC[10] SDA LIN2TX ADC1_S[2]	SIUL eMIOS_0 I <sup>2</sup> C_0 LINFlex_2 ADC_1	I/O I/O I/O O I	J	Tristate	74	107	131	B16
PA[11]	PCR[11]	AF0 AF1 AF2 AF3 — — —	GPIO[11] E0UC[11] SCL — EIRQ[16] LIN2RX ADC1_S[3]	SIUL eMIOS_0 I <sup>2</sup> C_0 — SIUL LINFlex_2 ADC_1	I/O I/O I/O — I I I	J	Tristate	75	108	132	B15
PA[12]	PCR[12]	AF0 AF1 AF2 AF3 — —	GPIO[12] — E0UC[28] CS3_1 EIRQ[17] SIN_0	SIUL — eMIOS_0 DSPI_1 SIUL DSPI_0	I/O — I/O O I I	S	Tristate	31	45	53	T7

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PA[13]	PCR[13]	AF0 AF1 AF2 AF3	GPIO[13] SOUT_0 E0UC[29] —	SIUL DSPI_0 eMIOS_0 —	I/O O I/O —	M	Tristate	30	44	52	R7
PA[14]	PCR[14]	AF0 AF1 AF2 AF3 —	GPIO[14] SCK_0 CS0_0 E0UC[0] EIRQ[4]	SIUL DSPI_0 DSPI_0 eMIOS_0 SIUL	I/O I/O I/O I/O I	M	Tristate	28	42	50	P6
PA[15]	PCR[15]	AF0 AF1 AF2 AF3 —	GPIO[15] CS0_0 SCK_0 E0UC[1] WKPU[10](5)	SIUL DSPI_0 DSPI_0 eMIOS_0 WKPU	I/O I/O I/O I/O I	M	Tristate	27	40	48	R6
Port B											
PB[0]	PCR[16]	AF0 AF1 AF2 AF3	GPIO[16] CAN0TX E0UC[30] LIN0TX	SIUL FlexCAN_0 eMIOS_0 LINFlex_0	I/O O I/O O	M	Tristate	23	31	39	N3
PB[1]	PCR[17]	AF0 AF1 AF2 AF3 — — —	GPIO[17] — E0UC[31] — WKPU[4](5) CAN0RX LIN0RX	SIUL — eMIOS_0 — WKPU FlexCAN_0 LINFlex_0	I/O — I/O — I I I	S	Tristate	24	32	40	N1
PB[2]	PCR[18]	AF0 AF1 AF2 AF3	GPIO[18] LIN0TX SDA E0UC[30]	SIUL LINFlex_0 I <sup>2</sup> C_0 eMIOS_0	I/O O I/O I/O	M	Tristate	100	144	176	B2



Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PB[3]	PCR[19]	AF0 AF1 AF2 AF3 — —	GPIO[19] E0UC[31] SCL — WKPU[11] <sup>(5)</sup> LIN0RX	SIUL eMIOS_0 I <sup>2</sup> C_0 — WKPU LINFlex_0	I/O I/O I/O — I I	S	Tristate	1	1	1	C3
PB[4]	PCR[20]	AF0 AF1 AF2 AF3 — — —	— — — — ADC0_P[0] ADC1_P[0] GPIO[20]	— — — — ADC_0 ADC_1 SIUL	— — — — I I I	I	Tristate	50	72	88	T16
PB[5]	PCR[21]	AF0 AF1 AF2 AF3 — — —	— — — — ADC0_P[1] ADC1_P[1] GPIO[21]	— — — — ADC_0 ADC_1 SIUL	— — — — I I I	I	Tristate	53	75	91	R16
PB[6]	PCR[22]	AF0 AF1 AF2 AF3 — — —	— — — — ADC0_P[2] ADC1_P[2] GPIO[22]	— — — — ADC_0 ADC_1 SIUL	— — — — I I I	I	Tristate	54	76	92	P15
PB[7]	PCR[23]	AF0 AF1 AF2 AF3 — — —	— — — — ADC0_P[3] ADC1_P[3] GPIO[23]	— — — — ADC_0 ADC_1 SIUL	— — — — I I I	I	Tristate	55	77	93	P16

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PB[8]	PCR[24]	AF0 AF1 AF2 AF3 — — — —	GPIO[24] — — — OSC32K_XTAL <sup>(8)</sup> WKPU[25] <sup>(5)</sup> ADC0_S[0] ADC1_S[4]	SIUL — — — OSC32K WKPU ADC_0 ADC_1	I — — — — I <sup>(9)</sup> I I	I	—	39	53	61	R9
PB[9]	PCR[25]	AF0 AF1 AF2 AF3 — — — —	GPIO[25] — — — OSC32K_EXTAL <sup>(8)</sup> WKPU[26] <sup>(5)</sup> ADC0_S[1] ADC1_S[5]	SIUL — — — OSC32K WKPU ADC_0 ADC_1	I — — — — I <sup>(9)</sup> I I	I	—	38	52	60	T9
PB[10]	PCR[26]	AF0 AF1 AF2 AF3 — — —	GPIO[26] — — — WKPU[8] <sup>(5)</sup> ADC0_S[2] ADC1_S[6]	SIUL — — — WKPU ADC_0 ADC_1	I/O — — — I I I	J	Tristate	40	54	62	P9
PB[11]	PCR[27]	AF0 AF1 AF2 AF3 —	GPIO[27] E0UC[3] — CS0_0 ADC0_S[3]	SIUL eMIOS_0 — DSPI_0 ADC_0	I/O I/O — I/O I	J	Tristate	—	—	97	N13
PB[12]	PCR[28]	AF0 AF1 AF2 AF3 —	GPIO[28] E0UC[4] — CS1_0 ADC0_X[0]	SIUL eMIOS_0 — DSPI_0 ADC_0	I/O I/O — O I	J	Tristate	61	83	101	M16

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PB[13]	PCR[29]	AF0	GPIO[29]	SIUL	I/O	J	Tristate	63	85	103	M13
		AF1	E0UC[5]	eMIOS_0	I/O						
		AF2	—	—	—						
		AF3	CS2_0	DSPI_0	O						
		—	ADC0_X[1]	ADC_0	I						
PB[14]	PCR[30]	AF0	GPIO[30]	SIUL	I/O	J	Tristate	65	87	105	L16
		AF1	E0UC[6]	eMIOS_0	I/O						
		AF2	—	—	—						
		AF3	CS3_0	DSPI_0	O						
		—	ADC0_X[2]	ADC_0	I						
PB[15]	PCR[31]	AF0	GPIO[31]	SIUL	I/O	J	Tristate	67	89	107	L13
		AF1	E0UC[7]	eMIOS_0	I/O						
		AF2	—	—	—						
		AF3	CS4_0	DSPI_0	O						
		—	ADC0_X[3]	ADC_0	I						
Port C											
PC[0](10)	PCR[32]	AF0	GPIO[32]	SIUL	I/O	M	Input, weak pull-up	87	126	154	A8
		AF1	—	—	—						
		AF2	TDI	JTAGC	I						
		AF3	—	—	—						
PC[1](10)	PCR[33]	AF0	GPIO[33]	SIUL	I/O	F(11)	Tristate	82	121	149	C9
		AF1	—	—	—						
		AF2	TDO	JTAGC	O						
		AF3	—	—	—						
PC[2]	PCR[34]	AF0	GPIO[34]	SIUL	I/O	M	Tristate	78	117	145	A11
		AF1	SCK_1	DSPI_1	I/O						
		AF2	CAN4TX	FlexCAN_4	O						
		AF3	DEBUG[0]	4	O						
		—	EIRQ[5]	SSCM	O						
		—	—	SIUL	I						

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PC[3]	PCR[35]	AF0 AF1 AF2 AF3 — — —	GPIO[35] CS0_1 MA[0] DEBUG[1] EIRQ[6] CAN1RX CAN4RX	SIUL DSPI_1 ADC_0 SSCM SIUL FlexCAN_1 FlexCAN_4	I/O I/O O O I I I	S	Tristate	77	116	144	B11
PC[4]	PCR[36]	AF0 AF1 AF2 AF3 — — —	GPIO[36] E1UC[31] — DEBUG[2] EIRQ[18] SIN_1 CAN3RX	SIUL eMIOS_1 — SSCM SIUL DSPI_1 FlexCAN_3	I/O I/O — O I I I	M	Tristate	92	131	159	B7
PC[5]	PCR[37]	AF0 AF1 AF2 AF3 —	GPIO[37] SOUT_1 CAN3TX DEBUG[3] EIRQ[7]	SIUL DSPI_1 FlexCAN_3 SSCM SIUL	I/O O O O I	M	Tristate	91	130	158	A7
PC[6]	PCR[38]	AF0 AF1 AF2 AF3	GPIO[38] LIN1TX E1UC[28] DEBUG[4]	SIUL LINFlex_1 eMIOS_1 SSCM	I/O O I/O O	S	Tristate	25	36	44	R2
PC[7]	PCR[39]	AF0 AF1 AF2 AF3 — —	GPIO[39] — E1UC[29] DEBUG[5] LIN1RX WKPU[12] <sup>(5)</sup>	SIUL — eMIOS_1 SSCM LINFlex_1 WKPU	I/O — I/O O I I	S	Tristate	26	37	45	P3
PC[8]	PCR[40]	AF0 AF1 AF2 AF3	GPIO[40] LIN2TX E0UC[3] DEBUG[6]	SIUL LINFlex_2 eMIOS_0 SSCM	I/O O I/O O	S	Tristate	99	143	175	A1

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PC[9]	PCR[41]	AF0 AF1 AF2 AF3 — —	GPIO[41] — E0UC[7] DEBUG[7] WKPU[13] <sup>(5)</sup> LIN2RX	SIUL — eMIOS_0 SSCM WKPU LINFlex_2	I/O — I/O O I I	S	Tristate	2	2	2	B1
PC[10]	PCR[42]	AF0 AF1 AF2 AF3	GPIO[42] CAN1TX CAN4TX MA[1]	SIUL FlexCAN_1 FlexCAN_4 ADC_0	I/O O O O	M	Tristate	22	28	36	M3
PC[11]	PCR[43]	AF0 AF1 AF2 AF3 — — —	GPIO[43] — — MA[2] WKPU[5] <sup>(5)</sup> CAN1RX CAN4RX	SIUL — — ADC_0 WKPU FlexCAN_1 FlexCAN_4	I/O — — O I I I	S	Tristate	21	27	35	M4
PC[12]	PCR[44]	AF0 AF1 AF2 AF3 — —	GPIO[44] E0UC[12] — — EIRQ[19] SIN_2	SIUL eMIOS_0 — — SIUL DSPI_2	I/O I/O — — I I	M	Tristate	97	141	173	B4
PC[13]	PCR[45]	AF0 AF1 AF2 AF3	GPIO[45] E0UC[13] SOUT_2 —	SIUL eMIOS_0 DSPI_2 —	I/O I/O O —	S	Tristate	98	142	174	A2
PC[14]	PCR[46]	AF0 AF1 AF2 AF3 —	GPIO[46] E0UC[14] SCK_2 — EIRQ[8]	SIUL eMIOS_0 DSPI_2 — SIUL	I/O I/O I/O — I	S	Tristate	3	3	3	C1

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PC[15]	PCR[47]	AF0 AF1 AF2 AF3 —	GPIO[47] E0UC[15] CS0_2 — EIRQ[20]	SIUL eMIOS_0 DSPI_2 — SIUL	I/O I/O I/O — I	M	Tristate	4	4	4	D3
Port D											
PD[0]	PCR[48]	AF0 AF1 AF2 AF3 — — —	GPIO[48] — — — WKPU[27](5) ADC0_P[4] ADC1_P[4]	SIUL — — — WKPU ADC_0 ADC_1	I — — — I I I	I	Tristate	41	63	77	P12
PD[1]	PCR[49]	AF0 AF1 AF2 AF3 — — —	GPIO[49] — — — WKPU[28](5) ADC0_P[5] ADC1_P[5]	SIUL — — — WKPU ADC_0 ADC_1	I — — — I I I	I	Tristate	42	64	78	T12
PD[2]	PCR[50]	AF0 AF1 AF2 AF3 — —	GPIO[50] — — — ADC0_P[6] ADC1_P[6]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	43	65	79	R12
PD[3]	PCR[51]	AF0 AF1 AF2 AF3 — —	GPIO[51] — — — ADC0_P[7] ADC1_P[7]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	44	66	80	P13

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PD[4]	PCR[52]	AF0 AF1 AF2 AF3 — —	GPIO[52] — — — ADC0_P[8] ADC1_P[8]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	45	67	81	R13
PD[5]	PCR[53]	AF0 AF1 AF2 AF3 — —	GPIO[53] — — — ADC0_P[9] ADC1_P[9]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	46	68	82	T13
PD[6]	PCR[54]	AF0 AF1 AF2 AF3 — —	GPIO[54] — — — ADC0_P[10] ADC1_P[10]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	47	69	83	T14
PD[7]	PCR[55]	AF0 AF1 AF2 AF3 — —	GPIO[55] — — — ADC0_P[11] ADC1_P[11]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	48	70	84	R14
PD[8]	PCR[56]	AF0 AF1 AF2 AF3 — —	GPIO[56] — — — ADC0_P[12] ADC1_P[12]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	49	71	87	T15
PD[9]	PCR[57]	AF0 AF1 AF2 AF3 — —	GPIO[57] — — — ADC0_P[13] ADC1_P[13]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	56	78	94	N15

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PD[10]	PCR[58]	AF0 AF1 AF2 AF3 — —	GPIO[58] — — — ADC0_P[14] ADC1_P[14]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	57	79	95	N14
PD[11]	PCR[59]	AF0 AF1 AF2 AF3 — —	GPIO[59] — — — ADC0_P[15] ADC1_P[15]	SIUL — — — ADC_0 ADC_1	I — — — I I	I	Tristate	58	80	96	N16
PD[12]	PCR[60]	AF0 AF1 AF2 AF3 —	GPIO[60] CS5_0 E0UC[24] — ADC0_S[4]	SIUL DSPI_0 eMIOS_0 — ADC_0	I/O O I/O — I	J	Tristate	—	—	100	M15
PD[13]	PCR[61]	AF0 AF1 AF2 AF3 —	GPIO[61] CS0_1 E0UC[25] — ADC0_S[5]	SIUL DSPI_1 eMIOS_0 — ADC_0	I/O I/O I/O — I	J	Tristate	62	84	102	M14
PD[14]	PCR[62]	AF0 AF1 AF2 AF3 —	GPIO[62] CS1_1 E0UC[26] — ADC0_S[6]	SIUL DSPI_1 eMIOS_0 — ADC_0	I/O O I/O — I	J	Tristate	64	86	104	L15
PD[15]	PCR[63]	AF0 AF1 AF2 AF3 —	GPIO[63] CS2_1 E0UC[27] — ADC0_S[7]	SIUL DSPI_1 eMIOS_0 — ADC_0	I/O O I/O — I	J	Tristate	66	88	106	L14
Port E											



Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PE[0]	PCR[64]	AF0 AF1 AF2 AF3 — —	GPIO[64] E0UC[16] — — WKPU[6](5) CAN5RX	SIUL eMIOS_0 — — WKPU FlexCAN_5	I/O I/O — — I I	S	Tristate	6	10	18	F1
PE[1]	PCR[65]	AF0 AF1 AF2 AF3	GPIO[65] E0UC[17] CAN5TX —	SIUL eMIOS_0 FlexCAN_5 —	I/O I/O O —	M	Tristate	8	12	20	F4
PE[2]	PCR[66]	AF0 AF1 AF2 AF3 — —	GPIO[66] E0UC[18] — — EIRQ[21] SIN_1	SIUL eMIOS_0 — — SIUL DSPI_1	I/O I/O — — I I	M	Tristate	89	128	156	D7
PE[3]	PCR[67]	AF0 AF1 AF2 AF3	GPIO[67] E0UC[19] SOUT_1 —	SIUL eMIOS_0 DSPI_1 —	I/O I/O O —	M	Tristate	90	129	157	C7
PE[4]	PCR[68]	AF0 AF1 AF2 AF3 —	GPIO[68] E0UC[20] SCK_1 — EIRQ[9]	SIUL eMIOS_0 DSPI_1 — SIUL	I/O I/O I/O — I	M	Tristate	93	132	160	D6
PE[5]	PCR[69]	AF0 AF1 AF2 AF3	GPIO[69] E0UC[21] CS0_1 MA[2]	SIUL eMIOS_0 DSPI_1 ADC_0	I/O I/O I/O O	M	Tristate	94	133	161	C6
PE[6]	PCR[70]	AF0 AF1 AF2 AF3 —	GPIO[70] E0UC[22] CS3_0 MA[1] EIRQ[22]	SIUL eMIOS_0 DSPI_0 ADC_0 SIUL	I/O I/O O O I	M	Tristate	95	139	167	B5

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PE[7]	PCR[71]	AF0 AF1 AF2 AF3 —	GPIO[71] E0UC[23] CS2_0 MA[0] EIRQ[23]	SIUL eMIOS_0 DSPI_0 ADC_0 SIUL	I/O I/O O O I	M	Tristate	96	140	168	C4
PE[8]	PCR[72]	AF0 AF1 AF2 AF3	GPIO[72] CAN2TX E0UC[22] CAN3TX	SIUL FlexCAN_2 eMIOS_0 FlexCAN_3	I/O O I/O O	M	Tristate	9	13	21	G2
PE[9]	PCR[73]	AF0 AF1 AF2 AF3 — — —	GPIO[73] — E0UC[23] — WKPU[7](5) CAN2RX CAN3RX	SIUL — eMIOS_0 — WKPU FlexCAN_2 FlexCAN_3	I/O — I/O — I I I	S	Tristate	10	14	22	G1
PE[10]	PCR[74]	AF0 AF1 AF2 AF3 —	GPIO[74] LIN3TX CS3_1 E1UC[30] EIRQ[10]	SIUL LINFlex_3 DSPI_1 eMIOS_1 SIUL	I/O O O I/O I	S	Tristate	11	15	23	G3
PE[11]	PCR[75]	AF0 AF1 AF2 AF3 — —	GPIO[75] E0UC[24] CS4_1 — LIN3RX WKPU[14](5)	SIUL eMIOS_0 DSPI_1 — LINFlex_3 WKPU	I/O I/O O — I I	S	Tristate	13	17	25	H2
PE[12]	PCR[76]	AF0 AF1 AF2 AF3 — — —	GPIO[76] — E1UC[19](12) — EIRQ[11] SIN_2 ADC1_S[7]	SIUL — eMIOS_1 — SIUL DSPI_2 ADC_1	I/O — I/O — I I I	J	Tristate	76	109	133	C14

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PE[13]	PCR[77]	AF0 AF1 AF2 AF3	GPIO[77] SOUT_2 E1UC[20] —	SIUL DSPI_2 eMIOS_1 —	I/O O I/O —	S	Tristate	—	103	127	D15
PE[14]	PCR[78]	AF0 AF1 AF2 AF3 —	GPIO[78] SCK_2 E1UC[21] — EIRQ[12]	SIUL DSPI_2 eMIOS_1 — SIUL	I/O I/O I/O — I	S	Tristate	—	112	136	C13
PE[15]	PCR[79]	AF0 AF1 AF2 AF3	GPIO[79] CS0_2 E1UC[22] —	SIUL DSPI_2 eMIOS_1 —	I/O I/O I/O —	M	Tristate	—	113	137	A13
Port F											
PF[0]	PCR[80]	AF0 AF1 AF2 AF3 —	GPIO[80] E0UC[10] CS3_1 — ADC0_S[8]	SIUL eMIOS_0 DSPI_1 — ADC_0	I/O I/O O — I	J	Tristate	—	55	63	N10
PF[1]	PCR[81]	AF0 AF1 AF2 AF3 —	GPIO[81] E0UC[11] CS4_1 — ADC0_S[9]	SIUL eMIOS_0 DSPI_1 — ADC_0	I/O I/O O — I	J	Tristate	—	56	64	P10
PF[2]	PCR[82]	AF0 AF1 AF2 AF3 —	GPIO[82] E0UC[12] CS0_2 — ADC0_S[10]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O I/O — I	J	Tristate	—	57	65	T10
PF[3]	PCR[83]	AF0 AF1 AF2 AF3 —	GPIO[83] E0UC[13] CS1_2 — ADC0_S[11]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O O — I	J	Tristate	—	58	66	R10

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PF[4]	PCR[84]	AF0 AF1 AF2 AF3 —	GPIO[84] E0UC[14] CS2_2 — ADC0_S[12]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O O — I	J	Tristate	—	59	67	N11
PF[5]	PCR[85]	AF0 AF1 AF2 AF3 —	GPIO[85] E0UC[22] CS3_2 — ADC0_S[13]	SIUL eMIOS_0 DSPI_2 — ADC_0	I/O I/O O — I	J	Tristate	—	60	68	P11
PF[6]	PCR[86]	AF0 AF1 AF2 AF3 —	GPIO[86] E0UC[23] CS1_1 — ADC0_S[14]	SIUL eMIOS_0 DSPI_1 — ADC_0	I/O I/O O — I	J	Tristate	—	61	69	T11
PF[7]	PCR[87]	AF0 AF1 AF2 AF3 —	GPIO[87] — CS2_1 — ADC0_S[15]	SIUL — DSPI_1 — ADC_0	I/O — O — I	J	Tristate	—	62	70	R11
PF[8]	PCR[88]	AF0 AF1 AF2 AF3	GPIO[88] CAN3TX CS4_0 CAN2TX	SIUL FlexCAN_3 DSPI_0 FlexCAN_2	I/O O O O	M	Tristate	—	34	42	P1
PF[9]	PCR[89]	AF0 AF1 AF2 AF3 — — —	GPIO[89] E1UC[1] CS5_0 — WKPU[22](5) CAN2RX CAN3RX	SIUL eMIOS_1 DSPI_0 — WKPU FlexCAN_2 FlexCAN_3	I/O I/O O — I I I	S	Tristate	—	33	41	N2

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PF[10]	PCR[90]	AF0	GPIO[90]	SIUL	I/O	M	Tristate	—	38	46	R3
		AF1	CS1_0	DSPI_0	O						
		AF2	LIN4TX	LINFlex_4	O						
		AF3	E1UC[2]	eMIOS_1	I/O						
PF[11]	PCR[91]	AF0	GPIO[91]	SIUL	I/O	S	Tristate	—	39	47	R4
		AF1	CS2_0	DSPI_0	O						
		AF2	E1UC[3]	eMIOS_1	I/O						
		AF3	—	—	—						
		—	WKPU[15](5)	WKPU	I						
—	LIN4RX	LINFlex_4	I								
PF[12]	PCR[92]	AF0	GPIO[92]	SIUL	I/O	M	Tristate	—	35	43	R1
		AF1	E1UC[25]	eMIOS_1	I/O						
		AF2	LIN5TX	LINFlex_5	O						
		AF3	—	—	—						
PF[13]	PCR[93]	AF0	GPIO[93]	SIUL	I/O	S	Tristate	—	41	49	T6
		AF1	E1UC[26]	eMIOS_1	I/O						
		AF2	—	—	—						
		AF3	—	—	—						
		—	WKPU[16](5)	WKPU	I						
—	LIN5RX	LINFlex_5	I								
PF[14]	PCR[94]	AF0	GPIO[94]	SIUL	I/O	M	Tristate	—	102	126	D14
		AF1	CAN4TX	FlexCAN_4	O						
		AF2	E1UC[27]	eMIOS_1	I/O						
		AF3	CAN1TX	FlexCAN_1	O						
PF[15]	PCR[95]	AF0	GPIO[95]	SIUL	I/O	S	Tristate	—	101	125	E15
		AF1	E1UC[4]	eMIOS_1	I/O						
		AF2	—	—	—						
		AF3	—	SIUL	—						
		—	EIRQ[13]	FlexCAN_1	I						
		—	CAN1RX	FlexCAN_1	I						
—	CAN4RX	FlexCAN_4	I								

Port G

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PG[0]	PCR[96]	AF0 AF1 AF2 AF3	GPIO[96] CAN5TX E1UC[23] —	SIUL FlexCAN_5 eMIOS_1 —	I/O O I/O —	M	Tristate	—	98	122	E14
PG[1]	PCR[97]	AF0 AF1 AF2 AF3 — —	GPIO[97] — E1UC[24] — EIRQ[14] CAN5RX	SIUL — eMIOS_1 — SIUL FlexCAN_5	I/O — I/O — I I	S	Tristate	—	97	121	E13
PG[2]	PCR[98]	AF0 AF1 AF2 AF3	GPIO[98] E1UC[11] SOUT_3 —	SIUL eMIOS_1 DSPI_3 —	I/O I/O O —	M	Tristate	—	8	16	E4
PG[3]	PCR[99]	AF0 AF1 AF2 AF3 —	GPIO[99] E1UC[12] CS0_3 — WKPU[17](5)	SIUL eMIOS_1 DSPI_3 — WKPU	I/O I/O I/O — I	S	Tristate	—	7	15	E3
PG[4]	PCR[100]	AF0 AF1 AF2 AF3	GPIO[100] E1UC[13] SCK_3 —	SIUL eMIOS_1 DSPI_3 —	I/O I/O I/O —	M	Tristate	—	6	14	E1
PG[5]	PCR[101]	AF0 AF1 AF2 AF3 — —	GPIO[101] E1UC[14] — — WKPU[18](5) SIN_3	SIUL eMIOS_1 — — WKPU DSPI_3	I/O I/O — — I I	S	Tristate	—	5	13	E2
PG[6]	PCR[102]	AF0 AF1 AF2 AF3	GPIO[102] E1UC[15] LIN6TX —	SIUL eMIOS_1 LINFlex_6 —	I/O I/O O —	M	Tristate	—	30	38	M2

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PG[7]	PCR[103]	AF0 AF1 AF2 AF3 — —	GPIO[103] E1UC[16] E1UC[30] — WKPU[20] <sup>(5)</sup> LIN6RX	SIUL eMIOS_1 eMIOS_1 — WKPU LINFlex_6	I/O I/O I/O — I I	S	Tristate	—	29	37	M1
PG[8]	PCR[104]	AF0 AF1 AF2 AF3 —	GPIO[104] E1UC[17] LIN7TX CS0_2 EIRQ[15]	SIUL eMIOS_1 LINFlex_7 DSPI_2 SIUL	I/O I/O O I/O I	S	Tristate	—	26	34	L2
PG[9]	PCR[105]	AF0 AF1 AF2 AF3 — —	GPIO[105] E1UC[18] — SCK_2 WKPU[21] <sup>(5)</sup> LIN7RX	SIUL eMIOS_1 — DSPI_2 WKPU LINFlex_7	I/O I/O — I/O I I	S	Tristate	—	25	33	L1
PG[10]	PCR[106]	AF0 AF1 AF2 AF3 —	GPIO[106] E0UC[24] E1UC[31] — SIN_4	SIUL eMIOS_0 eMIOS_1 — DSPI_4	I/O I/O I/O — I	S	Tristate	—	114	138	D13
PG[11]	PCR[107]	AF0 AF1 AF2 AF3	GPIO[107] E0UC[25] CS0_4 —	SIUL eMIOS_0 DSPI_4 —	I/O I/O I/O —	M	Tristate	—	115	139	B12
PG[12]	PCR[108]	AF0 AF1 AF2 AF3	GPIO[108] E0UC[26] SOUT_4 —	SIUL eMIOS_0 DSPI_4 —	I/O I/O O —	M	Tristate	—	92	116	K14
PG[13]	PCR[109]	AF0 AF1 AF2 AF3	GPIO[109] E0UC[27] SCK_4 —	SIUL eMIOS_0 DSPI_4 —	I/O I/O I/O —	M	Tristate	—	91	115	K16

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PG[14]	PCR[110]	AF0 AF1 AF2 AF3	GPIO[110] E1UC[0] LIN8TX —	SIUL eMIOS_1 LINFlex_8 —	I/O I/O O —	S	Tristate	—	110	134	B14
PG[15]	PCR[111]	AF0 AF1 AF2 AF3 —	GPIO[111] E1UC[1] — — LIN8RX	SIUL eMIOS_1 — — LINFlex_8	I/O I/O — — I	M	Tristate	—	111	135	B13
Port H											
PH[0]	PCR[112]	AF0 AF1 AF2 AF3 —	GPIO[112] E1UC[2] — — SIN_1	SIUL eMIOS_1 — — DSPI_1	I/O I/O — — I	M	Tristate	—	93	117	F13
PH[1]	PCR[113]	AF0 AF1 AF2 AF3	GPIO[113] E1UC[3] SOUT_1 —	SIUL eMIOS_1 DSPI_1 —	I/O I/O O —	M	Tristate	—	94	118	F14
PH[2]	PCR[114]	AF0 AF1 AF2 AF3	GPIO[114] E1UC[4] SCK_1 —	SIUL eMIOS_1 DSPI_1 —	I/O I/O I/O —	M	Tristate	—	95	119	F16
PH[3]	PCR[115]	AF0 AF1 AF2 AF3	GPIO[115] E1UC[5] CS0_1 —	SIUL eMIOS_1 DSPI_1 —	I/O I/O I/O —	M	Tristate	—	96	120	F15
PH[4]	PCR[116]	AF0 AF1 AF2 AF3	GPIO[116] E1UC[6] — —	SIUL eMIOS_1 — —	I/O I/O — —	M	Tristate	—	134	162	A6
PH[5]	PCR[117]	AF0 AF1 AF2 AF3	GPIO[117] E1UC[7] — —	SIUL eMIOS_1 — —	I/O I/O — —	S	Tristate	—	135	163	B6



Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PH[6]	PCR[118]	AF0 AF1 AF2 AF3	GPIO[118] E1UC[8] — MA[2]	SIUL eMIOS_1 — ADC_0	I/O I/O — O	M	Tristate	—	136	164	D5
PH[7]	PCR[119]	AF0 AF1 AF2 AF3	GPIO[119] E1UC[9] CS3_2 MA[1]	SIUL eMIOS_1 DSPI_2 ADC_0	I/O I/O O O	M	Tristate	—	137	165	C5
PH[8]	PCR[120]	AF0 AF1 AF2 AF3	GPIO[120] E1UC[10] CS2_2 MA[0]	SIUL eMIOS_1 DSPI_2 ADC_0	I/O I/O O O	M	Tristate	—	138	166	A5
PH[9] <sup>(10)</sup>	PCR[121]	AF0 AF1 AF2 AF3	GPIO[121] — TCK —	SIUL — JTAGC —	I/O — I —	S	Input, weak pull-up	88	127	155	B8
PH[10] <sup>(10)</sup>	PCR[122]	AF0 AF1 AF2 AF3	GPIO[122] — TMS —	SIUL — JTAGC —	I/O — I —	M	Input, weak pull-up	81	120	148	B9
PH[11]	PCR[123]	AF0 AF1 AF2 AF3	GPIO[123] SOUT_3 CS0_4 E1UC[5]	SIUL DSPI_3 DSPI_4 eMIOS_1	I/O O I/O I/O	M	Tristate	—	—	140	A14
PH[12]	PCR[124]	AF0 AF1 AF2 AF3	GPIO[124] SCK_3 CS1_4 E1UC[25]	SIUL DSPI_3 DSPI_4 eMIOS_1	I/O I/O O I/O	M	Tristate	—	—	141	D12
PH[13]	PCR[125]	AF0 AF1 AF2 AF3	GPIO[125] SOUT_4 CS0_3 E1UC[26]	SIUL DSPI_4 DSPI_3 eMIOS_1	I/O O I/O I/O	M	Tristate	—	—	9	B3
PH[14]	PCR[126]	AF0 AF1 AF2 AF3	GPIO[126] SCK_4 CS1_3 E1UC[27]	SIUL DSPI_4 DSPI_3 eMIOS_1	I/O I/O O I/O	M	Tristate	—	—	10	D1

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PH[15]	PCR[127]	AF0	GPIO[127]	SIUL	I/O	M	Tristate	—	—	8	A3
		AF1	SOUT_5	DSPI_5	O						
		AF2	—	—	—						
		AF3	E1UC[17]	eMIOS_1	I/O						
Port I											
PI[0]	PCR[128]	AF0	GPIO[128]	SIUL	I/O	S	Tristate	—	—	172	A9
		AF1	E0UC[28]	eMIOS_0	I/O						
		AF2	LIN8TX	LINFlex_8	O						
		AF3	—	—	—						
PI[1]	PCR[129]	AF0	GPIO[129]	SIUL	I/O	S	Tristate	—	—	171	A10
		AF1	E0UC[29]	eMIOS_0	I/O						
		AF2	—	—	—						
		AF3	—	—	—						
		—	WKPU[24] <sup>(5)</sup>	WKPU	I						
—	LIN8RX	LINFlex_8	I								
PI[2]	PCR[130]	AF0	GPIO[130]	SIUL	I/O	S	Tristate	—	—	170	B10
		AF1	E0UC[30]	eMIOS_0	I/O						
		AF2	LIN9TX	LINFlex_9	O						
		AF3	—	—	—						
PI[3]	PCR[131]	AF0	GPIO[131]	SIUL	I/O	S	Tristate	—	—	169	C10
		AF1	E0UC[31]	eMIOS_0	I/O						
		AF2	—	—	—						
		AF3	—	—	—						
		—	WKPU[23] <sup>5</sup>	WKPU	I						
—	LIN9RX	LINFlex_9	I								
PI[4]	PCR[132]	AF0	GPIO[132]	SIUL	I/O	S	Tristate	—	—	143	A12
		AF1	E1UC[28]	eMIOS_1	I/O						
		AF2	SOUT_4	DSPI_4	O						
		AF3	—	—	—						
PI[5]	PCR[133]	AF0	GPIO[133]	SIUL	I/O	S	Tristate	—	—	142	C12
		AF1	E1UC[29]	eMIOS_1	I/O						
		AF2	SCK_4	DSPI_4	I/O						
		AF3	—	—	—						

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PI[6]	PCR[134]	AF0 AF1 AF2 AF3	GPIO[134] E1UC[30] CS0_4 —	SIUL eMIOS_1 DSPI_4 —	I/O I/O I/O —	S	Tristate	—	—	11	D2
PI[7]	PCR[135]	AF0 AF1 AF2 AF3	GPIO[135] E1UC[31] CS1_4 —	SIUL eMIOS_1 DSPI_4 —	I/O I/O O —	S	Tristate	—	—	12	D3
PI[8]	PCR[136]	AF0 AF1 AF2 AF3 —	GPIO[136] — — — ADC0_S[16]	SIUL — — — ADC_0	I/O — — — I	J	Tristate	—	—	108	J13
PI[9]	PCR[137]	AF0 AF1 AF2 AF3 —	GPIO[137] — — — ADC0_S[17]	SIUL — — — ADC_0	I/O — — — I	J	Tristate	—	—	109	J14
PI[10]	PCR[138]	AF0 AF1 AF2 AF3 —	GPIO[138] — — — ADC0_S[18]	SIUL — — — ADC_0	I/O — — — I	J	Tristate	—	—	110	J15
PI[11]	PCR[139]	AF0 AF1 AF2 AF3 — —	GPIO[139] — — — ADC0_S[19] SIN_3	SIUL — — — ADC_0 DSPI_3	I/O — — — I I	J	Tristate	—	—	111	J16
PI[12]	PCR[140]	AF0 AF1 AF2 AF3 —	GPIO[140] CS0_3 — — ADC0_S[20]	SIUL DSPI_3 — — ADC_0	I/O I/O — — I	J	Tristate	—	—	112	G14

Table 10. Functional port pin descriptions (continued)

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PI[13]	PCR[141]	AF0 AF1 AF2 AF3 —	GPIO[141] CS1_3 — — ADC0_S[21]	SIUL DSPI_3 — — ADC_0	I/O O — — I	J	Tristate	—	—	113	G15
PI[14]	PCR[142]	AF0 AF1 AF2 AF3 — —	GPIO[142] — — — ADC0_S[22] SIN_4	SIUL — — — ADC_0 DSPI_4	I/O — — — I I	J	Tristate	—	—	76	R8
PI[15]	PCR[143]	AF0 AF1 AF2 AF3 —	GPIO[143] CS0_4 — — ADC0_S[23]	SIUL DSPI_4 — — ADC_0	I/O I/O — — I	J	Tristate	—	—	75	T8
Port J											
PJ[0]	PCR[144]	AF0 AF1 AF2 AF3 —	GPIO[144] CS1_4 — — ADC0_S[24]	SIUL DSPI_4 — — ADC_0	I/O O — — I	J	Tristate	—	—	74	N5
PJ[1]	PCR[145]	AF0 AF1 AF2 AF3 — —	GPIO[145] — — — ADC0_S[25] SIN_5	SIUL — — — ADC_0 DSPI_5	I/O — — — I I	J	Tristate	—	—	73	P5
PJ[2]	PCR[146]	AF0 AF1 AF2 AF3 —	GPIO[146] CS0_5 — — ADC0_S[26]	SIUL DSPI_5 — — ADC_0	I/O I/O — — I	J	Tristate	—	—	72	P4

**Table 10. Functional port pin descriptions (continued)**

Port pin	PCR register	Alternate function (1)	Function	Peripheral	I/O direction(2)	Pad type	RESET config.(3)	Pin number			
								LQFP 100	LQFP 144	LQFP 176	LBGA 208(4)
PJ[3]	PCR[147]	AF0	GPIO[147]	SIUL	I/O	J	Tristate	—	—	71	P2
		AF1	CS1_5	DSPI_5	O						
		AF2	—	—	—						
		AF3	—	—	—						
		—	ADC0_S[27]	ADC_0	I						
PJ[4]	PCR[148]	AF0	GPIO[148]	SIUL	I/O	M	Tristate	—	—	5	A4
		AF1	SCK_5	DSPI_5	I/O						
		AF2	E1UC[18]	eMIOS_1	I/O						
		AF3	—	—	—						
		—	—	—	—						

- Alternate functions are chosen by setting the values of the PCR.PA bitfields inside the SIUL module. PCR.PA = 00-> AF0; PCR.PA = 01 -> AF1; PCR.PA = 10 -> AF2; PCR.PA = 11-> AF3. This is intended to select the output functions; to use one of the input functions, the PCR.IBE bit must be written to '1', regardless of the values selected in the PCR.PA bitfields. For this reason, the value corresponding to an input only function is reported as "—".
- Multiple inputs are routed to all respective modules internally. The input of some modules must be configured by setting the values of the PSMIO.PADSELx bitfields inside the SIUL module.
- The RESET configuration applies during and after reset.
- LBGA208 available only as development package for Nexus2+
- All WKPU pins also support external interrupt capability. See the WKPU chapter for further details.
- NMI has higher priority than alternate function. When NMI is selected, the PCR.AF field is ignored.
- "Not applicable" because these functions are available only while the device is booting. Refer to the BAM information for details.
- Value of PCR.IBE bit must be 0
- This wakeup input cannot be used to exit STANDBY mode.
- Out of reset all the functional pins except PC[0:1] and PH[9:10] are available to the user as GPIO. PC[0:1] are available as JTAG pins (TDI and TDO respectively). PH[9:10] are available as JTAG pins (TCK and TMS respectively). It is up to the user to configure these pins as GPIO when needed.
- PC[1] is a fast/medium pad but is in medium configuration by default. This pad is in Alternate Function 2 mode after reset which has TDO functionality. The reset value of PCR.OBE is '1', but this setting has no impact as long as this pad stays in AF2 mode. After configuring this pad as GPIO (PCR.PA = 0), output buffer is enabled as reset value of PCR.OBE = 1.
- Not available in LQFP100 package

## 4.8 Nexus 2+ pins

In the LBGA208 package, eight additional debug pins are available (see [Table 11](#)).

Table 11. Nexus 2+ pin descriptions

Port pin	Function	I/O direction	Pad type	Function after reset	Pin number		
					LQFP 100	LQFP 144	LBGA 208 <sup>(1)</sup>
MCKO	Message clock out	O	F	—	—	—	T4
MDO0	Message data out 0	O	M	—	—	—	H15
MDO1	Message data out 1	O	M	—	—	—	H16
MDO2	Message data out 2	O	M	—	—	—	H14
MDO3	Message data out 3	O	M	—	—	—	H13
EVTI	Event in	I	M	Pull-up	—	—	K1
EVTO	Event out	O	M	—	—	—	L4
MSEO	Message start/end out	O	M	—	—	—	G16

1. LBGA208 available only as development package for Nexus2+

## 5 Microcontroller Boot

This chapter explains the process of booting the microcontroller. The following entities are involved in the boot process:

- [Section 5.2: Boot Assist Module \(BAM\)](#)
- [Section 5.3: System Status and Configuration Module \(SSCM\)](#)
- Flash memory boot sectors (see [Chapter 30: Flash Memory](#))
- Memory Management Unit (MMU)

### 5.1 Boot mechanism

This section describes the configuration required by the user, and the steps performed by the microcontroller, in order to achieve a successful boot from flash memory or serial download modes.

There are 2 external pins on the microcontroller that are latched during reset and used to determine whether the microcontroller will boot from flash memory or attempt a serial download via FlexCAN or LINFlex (RS232):

- FAB (Force Alternate Boot mode) on pin PA[9]
- ABS (Alternate Boot Select) on pin PA[8]

[Table 12](#) describes the configuration options.

**Table 12. Boot mode selection**

Mode	FAB pin (PA[9])	ABS pin (PA[8])
Flash memory boot (default mode)	0	X
Serial boot (LINFlex)	1	0
Serial boot (FlexCAN)	1	1

The microcontroller has a weak pull-down on PA[9] and a weak pull-up on PA[8]. This means that if nothing external is connected to these pins, the microcontroller will enter flash memory boot mode by default. In order to change the boot behavior, you should use external pullup or pulldown resistors on PA[9] and PA[8]. If there is any external circuitry connected to either pin, you must ensure that this does not interfere with the expected value applied to the pin at reset. Otherwise, the microcontroller may boot into an unexpected mode after reset.

The SSCM performs a lot of the automated boot activity including reading the latched value of the FAB (PA[9]) pin to determine whether to boot from flash memory or serial boot mode. This is illustrated in [Figure 7](#).

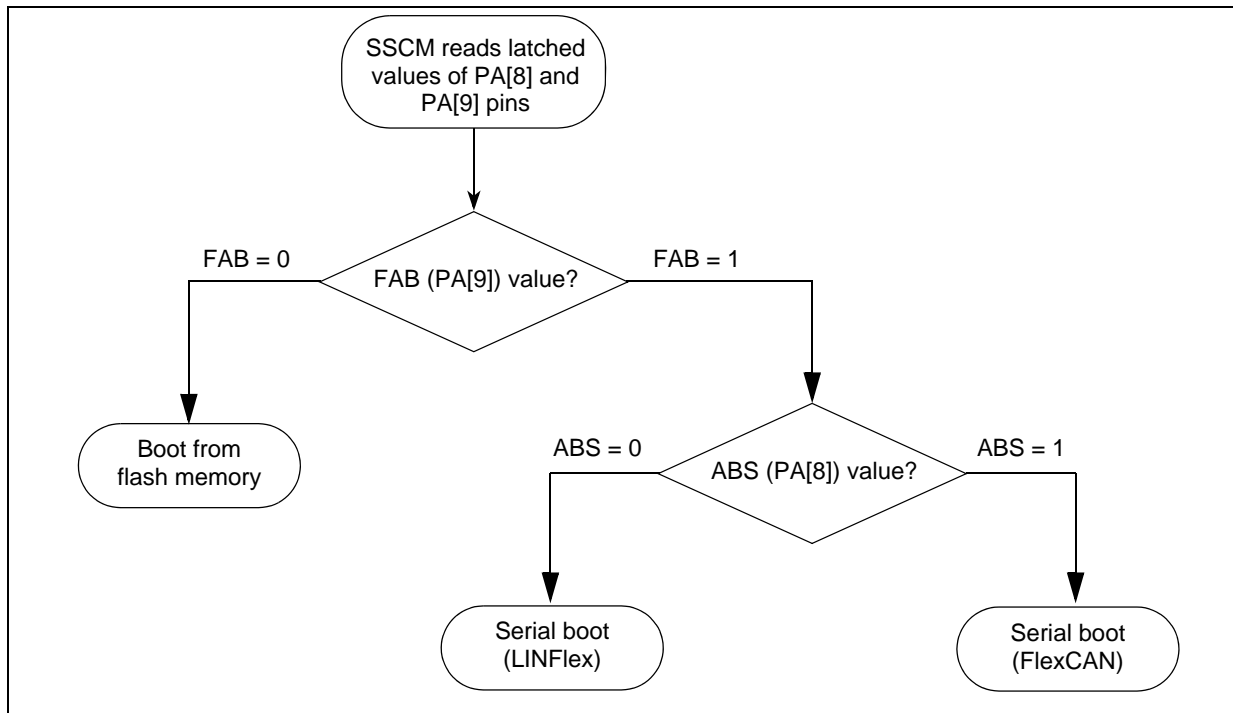


Figure 7. Boot mode selection

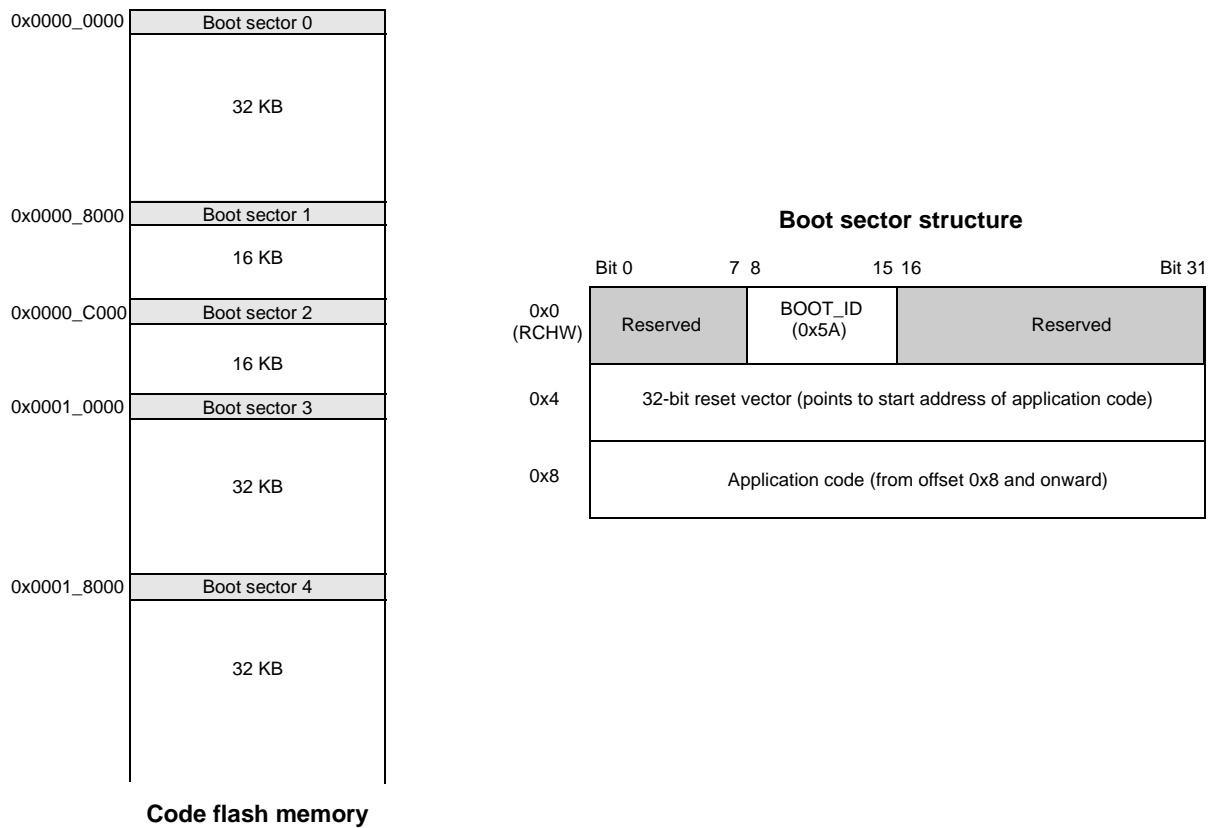
### 5.1.1 Flash memory boot

In order to successfully boot from flash memory, you must program two 32-bit fields into one of 5 possible boot blocks as detailed below. The entities to program are:

- 16-bit Reset Configuration Half Word (RCHW), which contains:
  - A BOOT\_ID field that must be correctly set to 0x5A in order to "validate" the boot sector
- 32-bit reset vector (this is the start address of the user code)

The location and structure of the boot sectors in flash memory are shown in [Figure 8](#).





**Figure 8. Boot sector structure**

The RCHW fields are described in [Table 13](#).

**Table 13. RCHW field descriptions**

Field	Description
BOOT_ID	Boot identifier. If BOOT_ID = 0x5A, the boot sector is considered valid and bootable.

The SSCM performs a sequential search of each boot sector (starting at sector 0) for a valid BOOT\_ID within the RCHW. If a valid BOOT\_ID is found, the SSCM reads the boot vector address. If a valid BOOT\_ID is not found, the SSCM starts the process of putting the microcontroller into static mode.

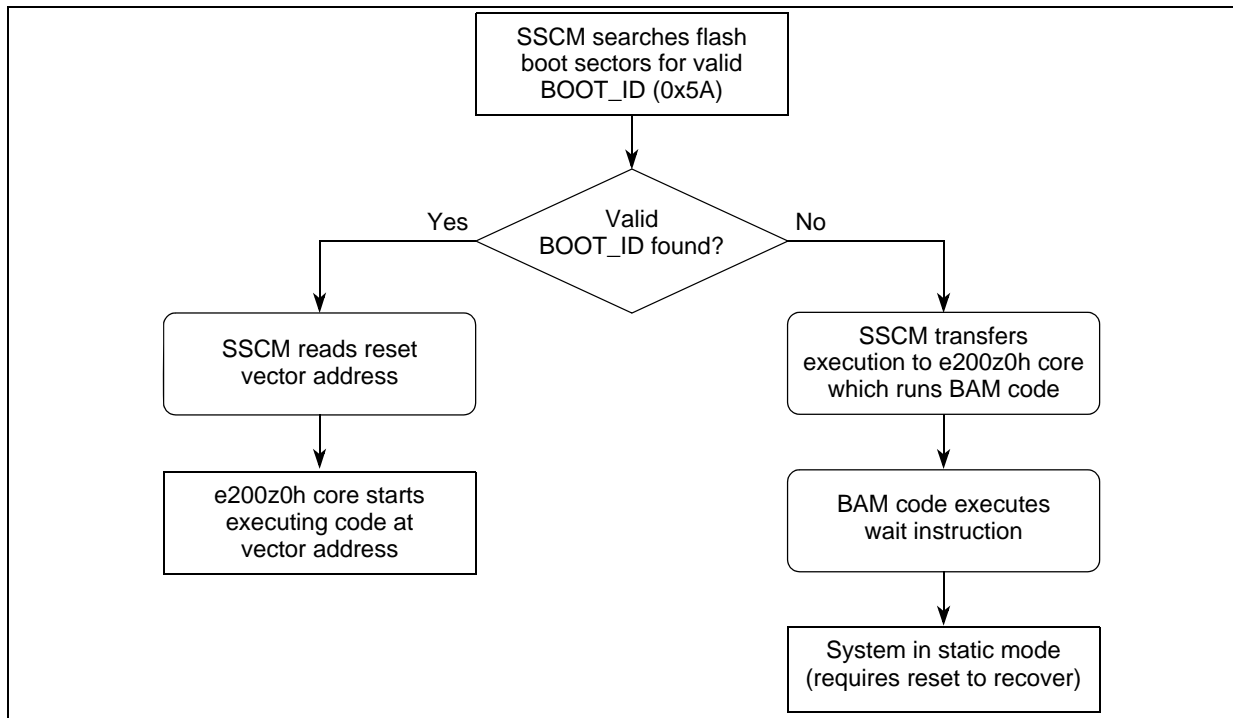
Finally, the SSCM sets the e200z0h core instruction pointer to the reset vector address and starts the core running.

**5.1.1.1 Static mode**

If no valid BOOT\_ID within the RCHW was found, the SSCM sets the CPU core instruction pointer to the BAM address and the core starts to execute the code to enter static mode as follows:

- The core executes the "wait" instruction which halts the core.

The sequence is illustrated in [Figure 9](#).



**Figure 9. Flash memory boot mode sequence**

**5.1.1.2 Alternate boot sectors**

Some applications require an alternate boot sector so that the main boot code can be erased and reprogrammed in the field. When an alternate boot is needed, you can create two bootable sectors:

- The valid boot sector located at the lowest address is the main boot sector.
- The valid boot sector located at the next available address is the alternate boot sector.

This scheme ensures that there is always one active boot sector even if the main boot sector is erased.

**5.1.2 Serial boot mode**

Serial boot provides a mechanism to download and then execute code into the microcontroller SRAM. Code may be downloaded using either FlexCAN or LINFlex (RS232). After the SSCM has detected that serial boot mode has been requested, execution is transferred to the BAM which handles all of the serial boot mode tasks. See [Section 5.2, Boot Assist Module \(BAM\)](#), for more details.

### 5.1.3 Censorship

Censorship can be enabled to protect the contents of the flash memory from being read or modified. In order to achieve this, the censorship mechanism controls access to the:

- JTAG / Nexus debug interface
- Serial boot mode (which could otherwise be used to download and execute code to query or modify the flash memory)

To re-gain access to the flash memory via JTAG or serial boot, a 64-bit password must be correctly entered.

**Caution:** When censorship has been enabled, the only way to regain access is with the password. If this is forgotten or not correctly configured, then there is no way back into the device.

There are two 64-bit values stored in the shadow flash which control the censorship (see [Table 437](#) for a full description):

- Nonvolatile Private Censorship Password registers, NVPWD0 and NVPWD1
- Nonvolatile System Censorship Control registers, NVSCC0 and NVSCC1

#### 5.1.3.1 Censorship password registers (NVPWD0 and NVPWD1)

The two private password registers combine to form a 64-bit password that should be programmed to a value known only by you. After factory test these registers are programmed as shown below:

- NVPWD0 = 0xFEED\_FACE
- NVPWD1 = 0xCAFE\_BEEF

This means that even if censorship was inadvertently enabled by writing to the censorship control registers, there is an opportunity to get back into the microcontroller using the default private password of 0xFEED\_FACE\_CAFE\_BEEF.

When configuring the private password, each half word (16-bit) must contain at least one "1" and one "0". Some examples of legal and illegal passwords are shown in [Table 14](#):

**Table 14. Examples of legal and illegal passwords**

Legal (valid) passwords	Illegal (invalid) passwords
0x0001_0001_0001_0001	0x0000_XXXX_XXXX_XXXX
0xFFFFE_FFFE_FFFE_FFFE	0xFFFFF_XXXX_XXXX_XXXX
0x1XXX_X2XX_XX4X_XXX8	

In uncensored devices it is possible to download code via LINFlex or FlexCAN (Serial Boot Mode) into internal SRAM even if the 64-bit private password stored in the flash and provided during the boot sequence is a password that does not conform to the password rules.

#### 5.1.3.2 Nonvolatile System Censorship Control registers (NVSCC0 and NVSCC1)

These registers are used together to define the censorship configuration. After factory test these registers are programmed as shown below which disables censorship:

- NVSCC0 = 0x55AA\_55AA
- NVSCC1 = 0x55AA\_55AA

Each 32-bit register is split into an upper and lower 16-bit field. The upper 16 bits (the SC field) are used to control serial boot mode censorship. The lower 16 bits (the CW field) are used to control flash memory boot censorship.

**Caution:** If the contents of the shadow flash memory are erased and the NVSCC0,1 registers are not re-programmed to a valid value, the microcontroller will be permanently censored with no way for you to regain access. A microcontroller in this state cannot be debugged or re-flashed.

### 5.1.3.3 Censorship configuration

The steps to configuring censorship are:

1. Define a valid 64-bit password that conforms to the password rules.
2. Using the table and flow charts below, decide what level of censorship you require and configure the NVSCC0,1 values.
3. Re-program the shadow flash memory and NVPWD0,1 and NVSCC0,1 registers with your new values. A POR is required before these will take effect.

**Caution:** If (NVSCC0 and NVSCC1 do not match) or (Either NVSCC0 or NVSCC1 is not set to 0x55AA) then the microcontroller will be permanently censored with no way to get back in.

Table 15 shows all the possible modes of censorship. The red shaded areas are to be avoided as these show the configuration for a device that is permanently locked out. If you wish to enable censorship with a private password there is only one valid configuration — to modify the CW field in both NVSCC0,1 registers so they match but do not equal 0x55AA. This will allow you to enter the private password in both serial and flash boot modes.

**Table 15. Censorship configuration and truth table**

Boot configuration		Serial censorship control word (NVSCCn[SC])	Censorship control word (NVSCCn[CW])	Internal flash memory state	Nexus state	Serial password	JTAG password
FAB pin state	Control options						
0 (flash memory boot)	Uncensored	0XXXX AND NVSCC0 == NVSCC1	0x55AA AND NVSCC0 == NVSCC1	Enabled	Enabled		N/A
	Private flash memory password and censored	0x55AA AND NVSCC0 == NVSCC1	!0x55AA AND NVSCC0 == NVSCC1	Enabled	Enabled with password		NVPWD1,0 (SSCM reads flash memory <sup>(1)</sup> )
	Censored with no password access (lockout)	!0x55AA OR NVSCC0 != NVSCC1	!0X55AA	Enabled	Disabled		N/A



**Table 15. Censorship configuration and truth table (continued)**

Boot configuration		Serial censorship control word (NVSCCn[SC])	Censorship control word (NVSCCn[CW])	Internal flash memory state	Nexus state	Serial password	JTAG password
FAB pin state	Control options						
1 (serial boot)	Private flash memory password and uncensored	0x55AA AND NVSCC0 == NVSCC1		Enabled	Enabled	NVPWD0,1 (BAM reads flash memory <sup>(1)</sup> )	
	Private flash memory password and censored	0x55AA AND NVSCC0 == NVSCC1	!0x55AA AND NVSCC0 == NVSCC1	Enabled	Disabled	NVPWD1,0 (SSCM reads flash memory <sup>(1)</sup> )	
	Public password and uncensored	!0x55AA AND NVSCC0 != NVSCC1	0X55AA AND NVSCC0 != NVSCC1	Enabled	Enabled	Public (0xFEED_FACE_CAFE_BEEF)	
	Public password and censored (lockout)	!0x55AA OR NVSCC0 != NVSCC1		Disabled	Disabled	Public (0xFEED_FACE_CAFE_BEEF)	

= Microcontroller permanently locked out  
 = Not applicable

1. When the SSCM reads the passwords from flash memory, the NVPWD0 and NVPWD1 password order is swapped, so you have to submit the 64-bit password as {NVPWD1, NVPWD0}.

The flow charts in [Figure 10](#) and [Figure 11](#) provide a way to quickly check what will happen with different configurations of the NVSCC0,1 registers as well as detailing the correct way to enter the serial password. In the password examples, assume the 64-bit password has been programmed into the shadow flash memory in the order {NVPWD0, NVPWD1} and has a value of 0x01234567\_89ABCDEF.

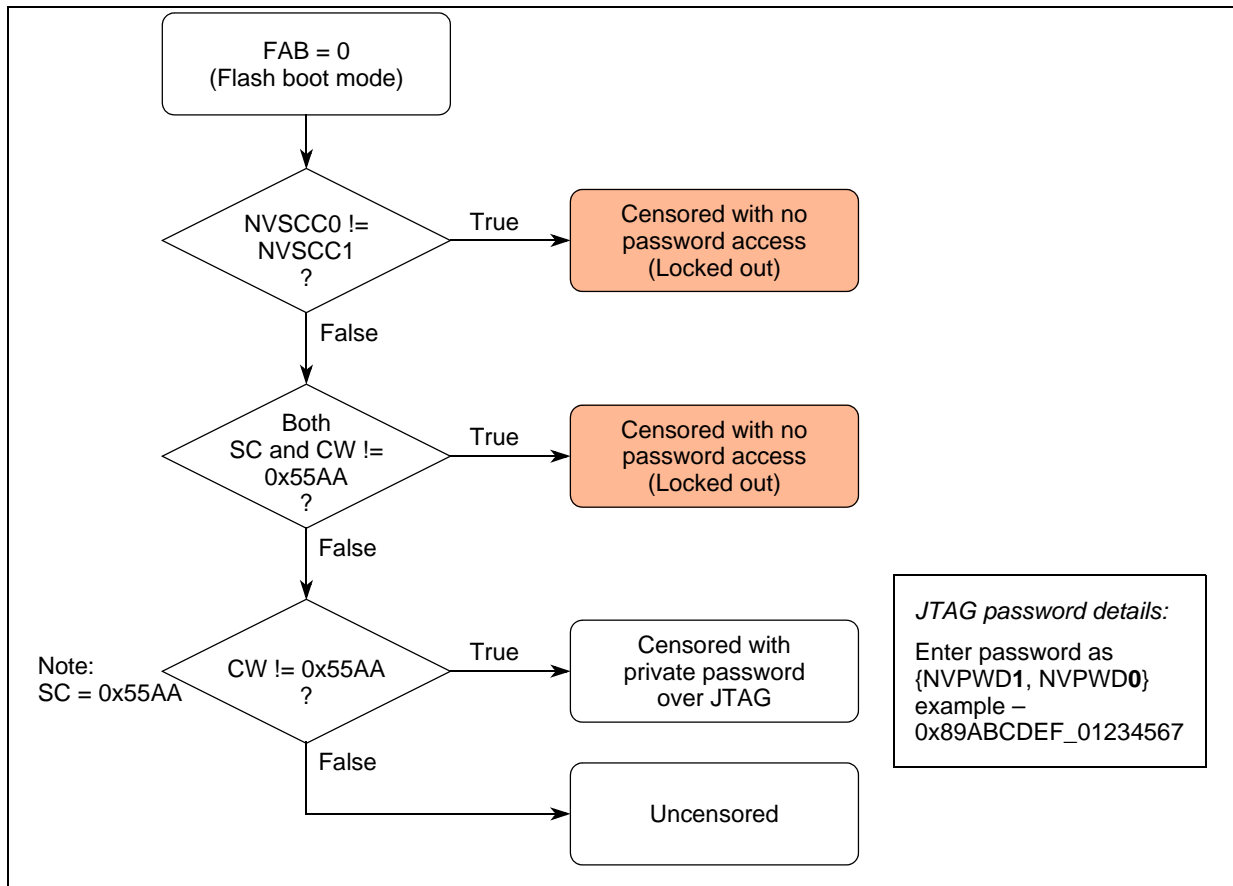


Figure 10. Censorship control in flash memory boot mode

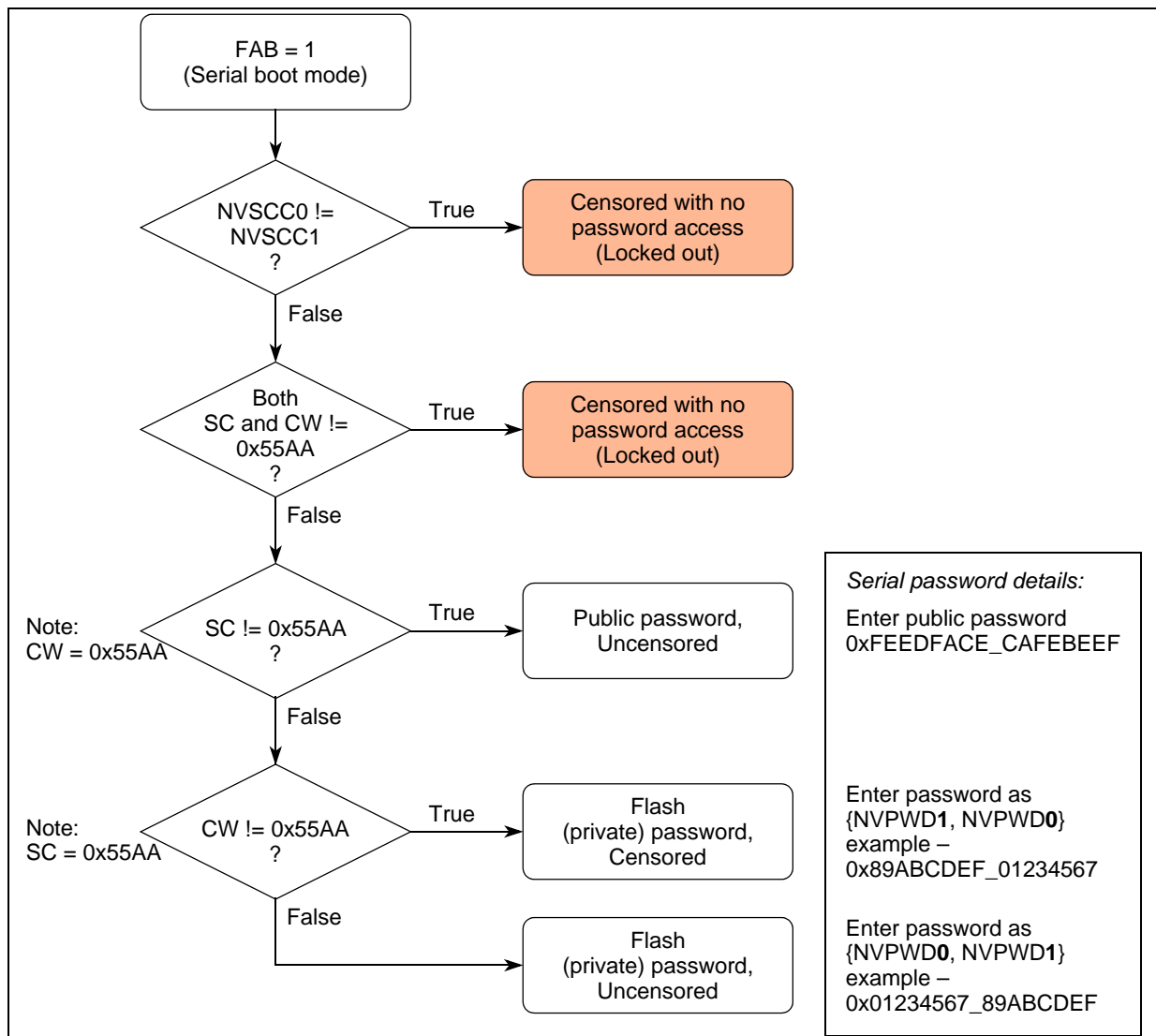


Figure 11. Censorship control in serial boot mode

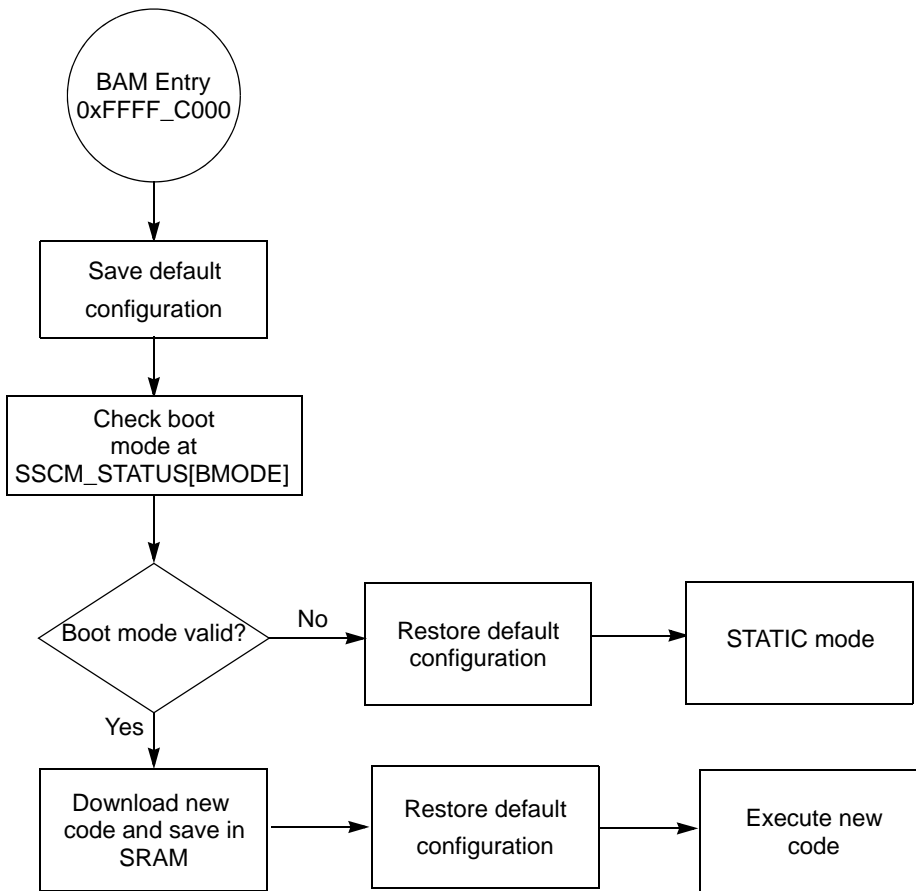
## 5.2 Boot Assist Module (BAM)

The BAM consist of a block of ROM at address 0xFFFF\_C000 containing VLE firmware. The BAM provides 2 main functions:

- Manages the serial download (FlexCAN or LINFlex protocols supported) including support for a serial password if censorship is enabled
- Places the microcontroller into static mode if flash memory boot mode is selected and a valid BOOT\_ID is not located in one of the boot sectors by the SSCM

### 5.2.1 BAM software flow

Figure 12 illustrates the BAM logic flow.



**Figure 12. BAM logic flow**

The initial (reset) device configuration is saved including the mode and clock configuration. This means that the serial download software running in the BAM can make changes to the modes and clocking and then restore these to the default values before running the newly downloaded application code from the SRAM.

The SSCM\_STATUS[BMODE] field indicates which boot mode is to be executed (see [Table 16](#)). This field is only updated during reset.

There are 2 conditions where the boot mode is not considered valid and the BAM pushes the microcontroller into static mode after restoring the default configuration:

- BMODE = 011 (flash memory boot mode). This means that the SSCM has been unable to find a valid BOOT\_ID in the boot sectors so has called the BAM
- BMODE = reserved

In static mode a wait instruction is executed to halt the core.

For the FlexCAN and LINFlex serial boot modes, the respective area of BAM code is executed to download the code to SRAM.



Table 16. SSCM\_STATUS[BMODE] values as used by BAM

BMODE value	Corresponding boot mode
000	Reserved
001	FlexCAN_0 serial boot loader
010	LINFlex_0 (RS232 /UART) serial boot loader
011	Flash memory boot mode
100–111	Reserved

After the code has been downloaded to SRAM, the BAM code restores the initial device configuration and then transfers execution to the start address of the downloaded code.

### 5.2.1.1 BAM resources

The BAM uses/initializes the following MCU resources:

- MC\_ME and MC\_CGM to initialize mode and clock sources
- FlexCAN\_0, LINFlex\_0 and the respective I/O pins when performing serial boot mode
- SSCM during password check
- SSCM to check the boot mode (see [Table 16](#))
- 4–16 MHz fast external crystal oscillator

The system clock is selected directly from the 4–16 MHz fast external crystal oscillator. Thus, the external oscillator frequency defines the baud rates used for serial download (see [Table 17](#)).

Table 17. Serial boot mode – baud rates

FXOSC frequency (MHz)	LINFlex baud rate (baud)	CAN bit rate (bit/s)
$f_{FXOSC}$	$f_{FXOSC}/833$	$f_{FXOSC}/40$
8	9600	200K
12	14400	300K
16	19200	400K

### 5.2.1.2 Download and execute the new code

From a high level perspective, the download protocol follows these steps:

1. Send the 64-bit password.
2. Send the start address, size of code to be downloaded (in bytes) and the VLE bit<sup>(f)</sup>.
3. Download the code.

Each step must be completed before the next step starts. After the download is complete (the specified number of bytes is downloaded), the code executes from the start address.

f. Since the device supports only VLE code and not Book E code, this flag is used only for backward compatibility.

The communication is done in half duplex manner, whereby the transmission from the host is followed by the microcontroller transmission mirroring the transmission back to the host:

- Host sends data to the microcontroller and waits for a response.
- MCU echoes to host the data received.
- Host verifies if echo is correct:
  - If data is correct, the host can continue to send data.
  - If data is not correct, the host stops transmission and the microcontroller enters static mode.

All multi-byte data structures are sent with MSB first.

A more detailed description of these steps follows.

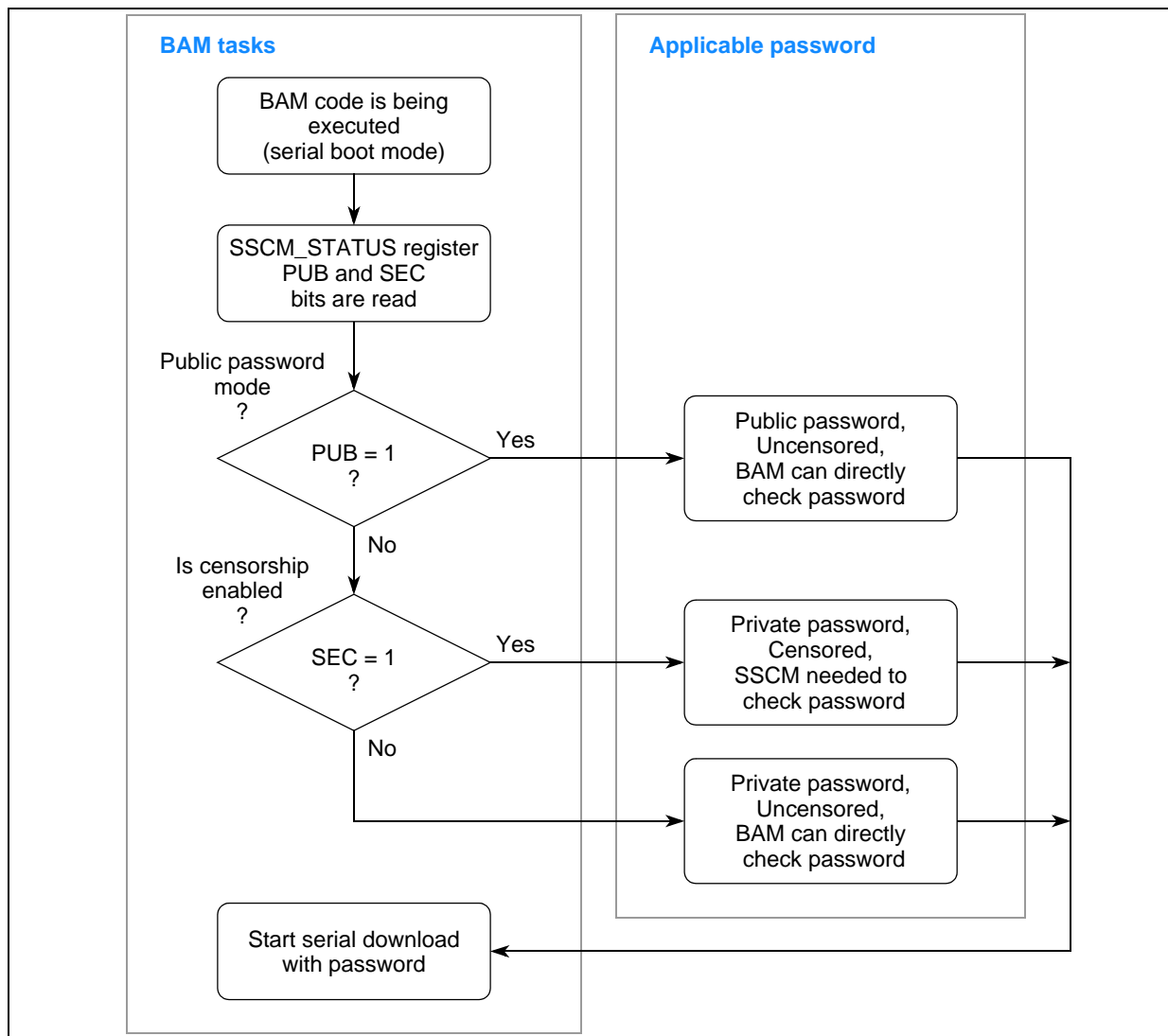
### 5.2.1.3 Censorship mode detection and serial password validation

Before the serial download can commence, the BAM code must determine which censorship mode the microcontroller is in and which password to use. It does this by reading the PUB and SEC fields in the SSCM Status Register (see [Section 5.3.4.1: System Status Register \(SSCM\\_STATUS\)](#)) as shown in [Table 18](#).

**Table 18. BAM censorship mode detection**

SSCM_STATUS register fields		Mode	Password comparison
PUB	SEC		
1	0	Uncensored, public password	0xFEED_FACE_CAFE_BEEF
0	0	Uncensored, private password	NVPWD0,1 from flash memory via BAM
0	1	Censored, private password	NVPWD1,0 from flash memory via SSCM

When censorship is enabled, the flash memory cannot be read by application code running in the BAM or in the SRAM. This means that the private password in the shadow flash memory cannot be read by the BAM code. In this case the SSCM is used to obtain the private password from the flash memory of the censored device. When the SSCM reads the private password it inverts the order of {NVPWD0, NVPWD1} so the password entered over the serial download needs to be {NVPWD1, NVPWD0}.



**Figure 13. BAM censorship mode detection**

The first thing to be downloaded is the 64-bit password. If the password does not match the stored password, then the BAM code pushes the microcontroller into static mode.

The way the password is compared with either the public or private password (depending on mode) varies depending on whether censorship is enabled as described in the following subsections.

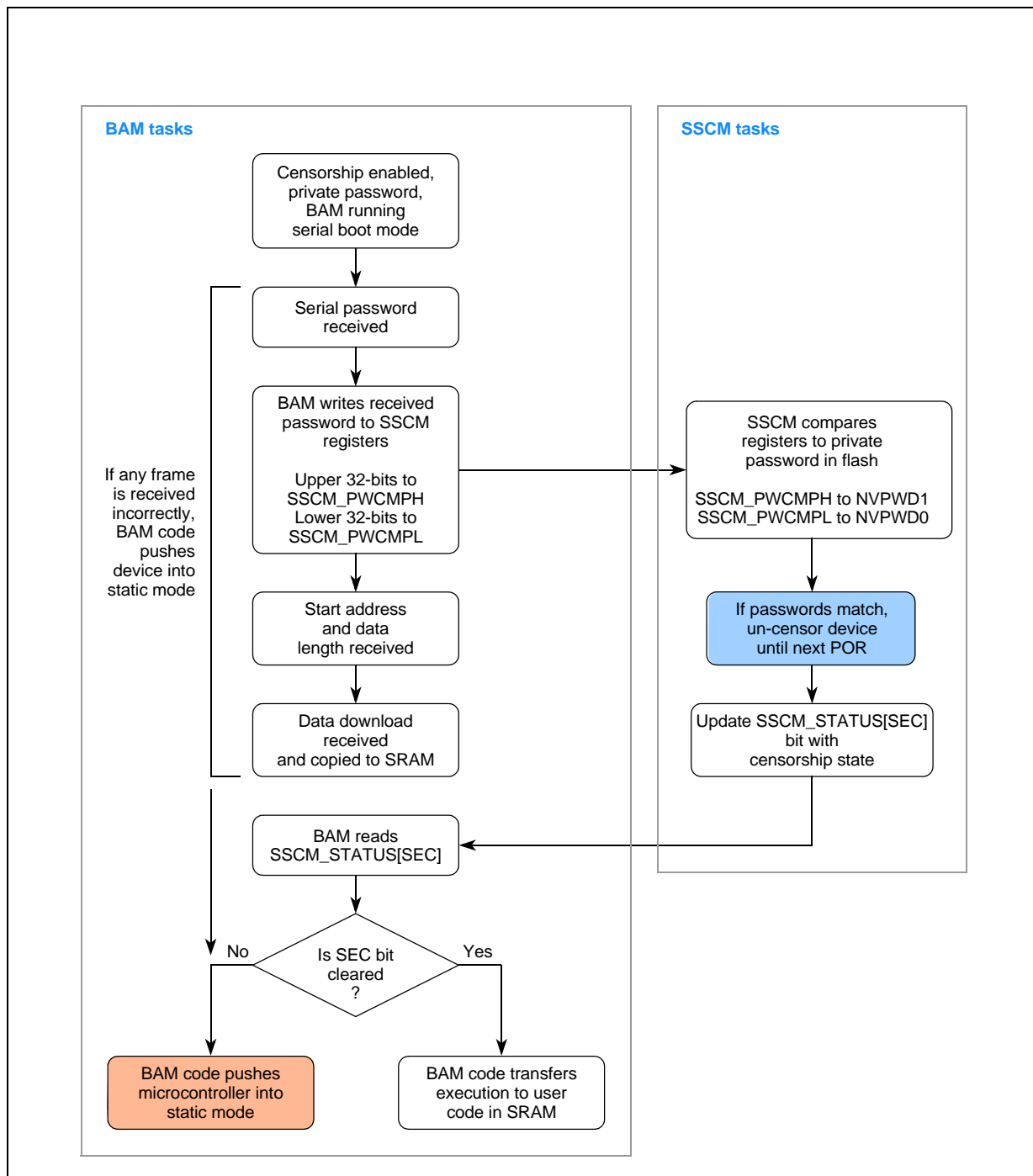
**5.2.1.3.1 Censorship disabled (private or public passwords):**

1. If the public password is used, the BAM code does a direct comparison between the serial password and 0xFEED\_FACE\_CAFE\_BEEF.
2. If the private password is used, the BAM code does a direct comparison between the serial password and the private password in flash memory, {NVPWD0, NVPWD1}.
3. If the password does not match, the BAM code immediately terminates the download and pushes the microcontroller into static mode.

### 5.2.1.3.2 Censorship enabled (private password)

1. Since the flash is secured, the SSCM is required to read the private password.
2. The BAM code writes the serial password to the SSCM\_PWCMPH and SSCM\_PWC MPL registers.
3. The BAM code then continues with the serial download (start address, data size and data) until all the data has been copied to the SRAM.
4. In the meantime the SSCM has compared the private password in flash with the serial download password the BAM code wrote into SSCM\_PWCMPH and SSCM\_PWC MPL.
5. If the SSCM obtains a match in the passwords, the censorship is temporarily disabled (until the next reset).
6. The SSCM updates the status of the security (SEC) bit to reflect whether the passwords matched (SEC = 0) or not (SEC = 1)
7. Finally, the BAM code reads SEC. If SEC = 0, execution is transferred to the code in the SRAM. If SEC = 1, the BAM code forces the microcontroller into static mode.

*Figure 14* shows this in more detail.



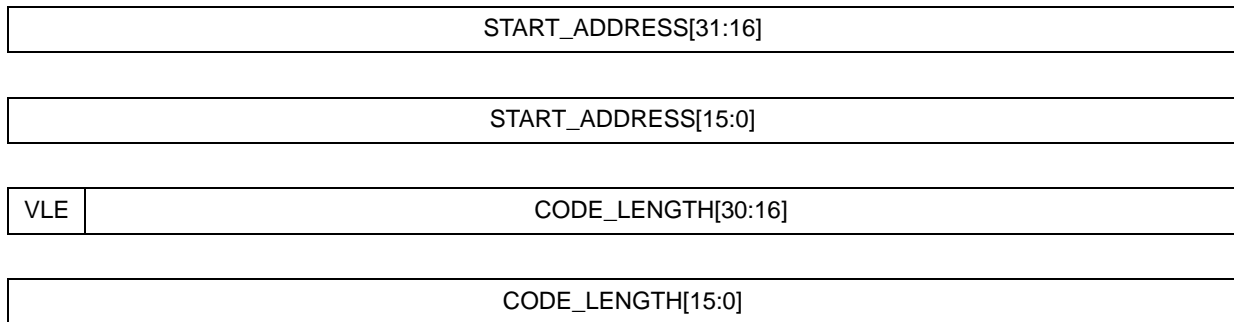
**Figure 14. BAM serial boot mode flow for censorship enabled and private password**

With LINFlex, any receive error will result in static mode. With FlexCAN, the host will re-transmit data if there has been no acknowledgment from the microcontroller. However there could be a situation where the receiver configuration has an error which would result in static mode entry.

*Note:* In a censored device booting with serial boot mode, it is possible to read the content of the four 32-bit flash memory locations that make up the boot sector. For example, if the RCHW is stored at address 0x0000\_0000, the reads at address 0x0000\_0000, 0x0000\_0004, 0x0000\_0008 and 0x0000\_000C will return a correct value. No other flash memory locations can be read.

**5.2.1.4 Download start address, VLE bit and code size**

The next 8 bytes received by the microcontroller contain a 32-bit Start Address, the VLE mode bit and a 31-bit code Length as shown in [Figure 15](#).



**Figure 15. Start address, VLE bit and download size in bytes**

The VLE bit (Variable Length Instruction) is used to indicate whether the code to be downloaded is Book VLE or Book III-E. This device family supports only VLE = 1; the bit is used for backward compatibility.

The Start Address defines where the received data will be stored and where the MCU will branch after the download is finished. The start address is 32-bit word aligned and the 2 least significant bits are ignored by the BAM code.

*Note:* The start address is configurable, but most not lie within the 0x4000\_0000 to 0x4000\_00FF address range.

The Length defines how many data bytes have to be loaded.

**5.2.1.5 Download data**

Each byte of data received is stored in the microcontroller’s SRAM, starting from the address specified in the previous protocol step.

The address increments until the number of bytes of data received matches the number of bytes specified by the code length.

Since the SRAM is protected by 32-bit wide Error Correction Code (ECC), the BAM code always writes bytes into SRAM grouped into 32-bit words. If the last byte received does not fall onto a 32-bit boundary, the BAM code fills any additional bytes with 0x0.

Since the ECC on the SRAM has not been initialized (except for the bytes of data that have just been downloaded), an additional dummy word of 0x0000\_0000 is written at the end of the downloaded data block to avoid any ECC errors during core prefetch.

### 5.2.1.6 Execute code

The BAM code waits for the last data byte to be received. If the operating mode is censored with a private password, then the BAM reads the SSCM status register to determine whether the serial password matched the private password. If there was a password match then the BAM code restores the initial configuration and transfers execution to the downloaded code start address in SRAM. If the passwords did not match, the BAM code forces a static mode entry.

*Note: The watchdog is disabled at the start of BAM code execution. In the case of an unexpected issue during BAM code execution, the microcontroller may be stalled and an external reset required to recover the microcontroller.*

## 5.2.2 LINFlex (RS232) boot

### 5.2.2.1 Configuration

Boot according to the LINFlex boot mode download protocol (see [Section 5.2.2.2: Protocol](#)) is performed by the LINFlex\_0 module in UART (RS232) mode. Pins used are:

- LIN0TX mapped on PB[2]
- LIN0RX mapped on PB[3]

Boot from LINFlex uses the system clock driven by the 4–16 MHz external crystal oscillator (FXOSC).

The LINFlex controller is configured to operate at a baud rate = system clock frequency/833, using an 8-bit data frame without parity bit and 1 stop bit.

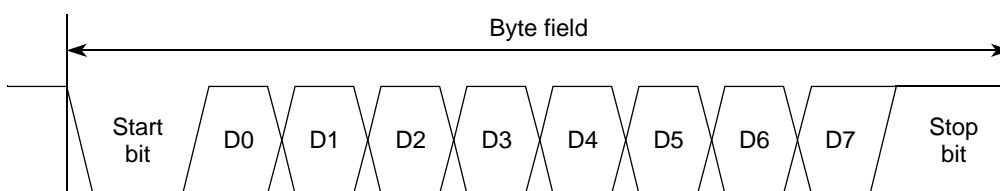


Figure 16. LINFlex bit timing in UART mode

### 5.2.2.2 Protocol

[Table 19](#) summarizes the protocol and BAM action during this boot mode.

Table 19. UART boot mode download protocol

Protocol step	Host sent message	BAM response message	Action
1	64-bit password (MSB first)	64-bit password	Password checked for validity and compared against stored password.
2	32-bit store address	32-bit store address	Load address is stored for future use.
3	VLE bit + 31-bit number of bytes (MSB first)	VLE bit + 31-bit number of bytes (MSB first)	Size of download are stored for future use. Verify if VLE bit is set to 1

**Table 19. UART boot mode download protocol (continued)**

Protocol step	Host sent message	BAM response message	Action
4	8 bits of raw binary data	8 bits of raw binary data	8-bit data are packed into a 32-bit word. This word is saved into SRAM starting from the "Load address". "Load address" increments until the number of data received and stored matches the size as specified in the previous step.
5	None	None	Branch to downloaded code

### 5.2.3 FlexCAN boot

#### 5.2.3.1 Configuration

Boot according to the FlexCAN boot mode download protocol (see [Section 5.2.3.2: Protocol](#)) is performed by the FlexCAN\_0 module. Pins used are:

- CAN0TX mapped on PB[0]
- CAN0RX mapped on PB[1]

*Note:* When the serial download via FlexCAN is selected and the device is part of a CAN network, the serial download may stop unexpectedly if there is any other traffic on the network. To avoid this situation, ensure that no other CAN device on the network is active during the serial download process.

Boot from FlexCAN uses the system clock driven by the 4–16 MHz fast external crystal oscillator.

The FlexCAN controller is configured to operate at a baud rate = system clock frequency/40 (see [Table 17](#) for examples of baud rate).

It uses the standard 11-bit identifier format detailed in FlexCAN 2.0A specification.

FlexCAN controller bit timing is programmed with 10 time quanta, and the sample point is 2 time quanta before the end, as shown in [Figure 17](#).



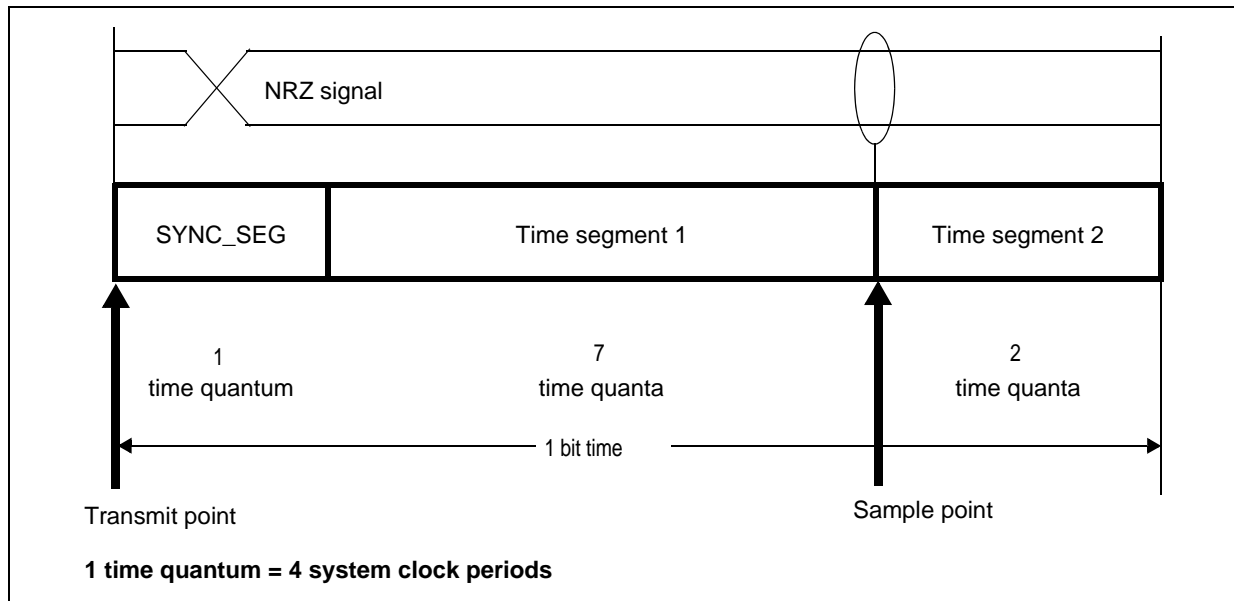


Figure 17. FlexCAN bit timing

5.2.3.2 Protocol

Table 20 summarizes the protocol and BAM action during this boot mode. All data are transmitted byte wise.

Table 20. FlexCAN boot mode download protocol

Protocol step	Host sent message	BAM response message	Action
1	CAN ID 0x011 + 64-bit password	CAN ID 0x001 + 64-bit password	Password checked for validity and compared against stored password
2	CAN ID 0x012 + 32-bit store address + VLE bit + 31-bit number of bytes	CAN ID 0x002 + 32-bit store address + VLE bit + 31-bit number of bytes	Load address is stored for future use. Size of download are stored for future use. Verify if VLE bit is set to 1
3	CAN ID 0x013 + 8 to 64 bits of raw binary data	CAN ID 0x003 + 8 to 64 bits of raw binary data	8-bit data are packed into 32-bit words. These words are saved into SRAM starting from the "Load address". "Load address" increments until the number of data received and stored matches the size as specified in the previous step.
5	None	None	Branch to downloaded code

## 5.3 System Status and Configuration Module (SSCM)

### 5.3.1 Introduction

The primary purpose of the SSCM is to provide information about the current state and configuration of the system that may be useful for configuring application software and for debug of the system.

On microcontrollers with a separate STANDBY power domain, the System Status block is part of that domain.

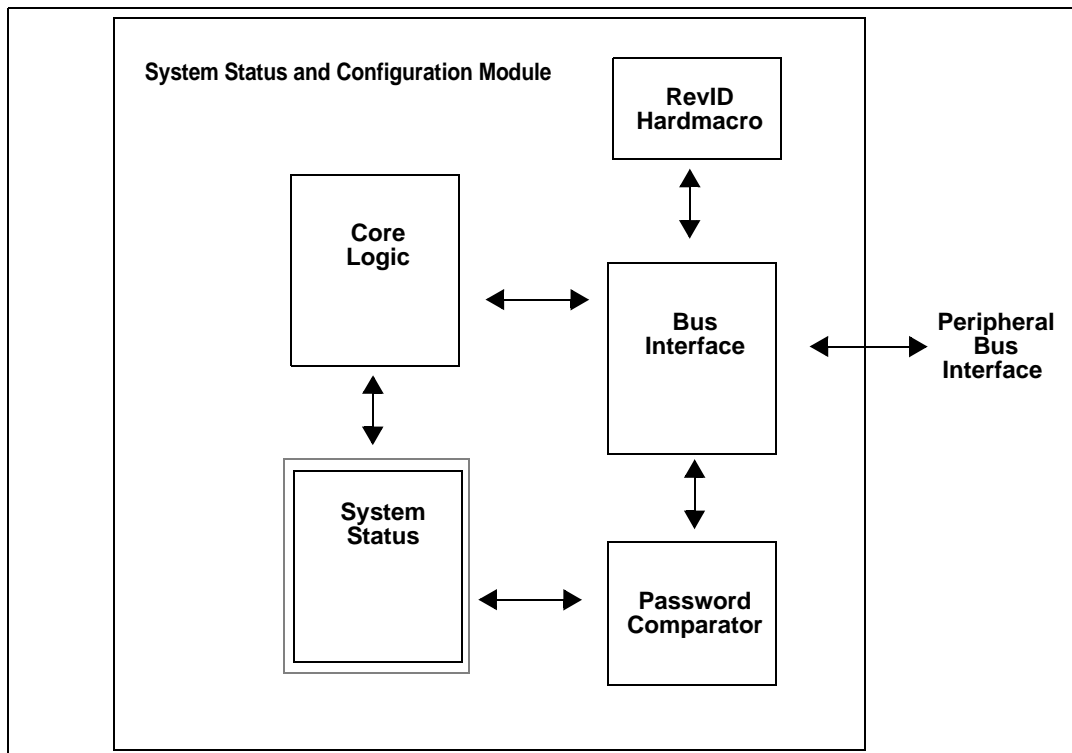


Figure 18. SSCM block diagram

### 5.3.2 Features

The SSCM includes these features:

- System Configuration and Status
  - Memory sizes/status
  - Microcontroller Mode and Security Status (including censorship and serial boot information)
  - Search Code Flash for bootable sector
  - Determine boot vector
- Device identification information (MCU ID Registers)
- Debug Status Port enable and selection
- Bus and peripheral abort enable/disable

### 5.3.3 Modes of operation

The SSCM operates identically in all system modes.

### 5.3.4 Memory map and register description

Table 21 shows the memory map for the SSCM. Note that all addresses are offsets; the absolute address may be calculated by adding the specified offset to the base address of the SSCM.

Table 21. SSCM memory map

Address offset	Register	Location
0x00	System Status Register (SSCM_STATUS)	<a href="#">on page 131</a>
0x02	System Memory Configuration Register (SSCM_MEMCONFIG)	<a href="#">on page 132</a>
0x04	Reserved	
0x06	Error Configuration (SSCM_ERROR)	<a href="#">on page 133</a>
0x08	Debug Status Port Register (SSCM_DEBUGPORT)	<a href="#">on page 134</a>
0x0A	Reserved	
0x0C	Password Comparison Register High Word (SSCM_PWCMPH)	<a href="#">on page 136</a>
0x10	Password Comparison Register Low Word (SSCM_PWC MPL)	<a href="#">on page 136</a>

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the SSCM\_STATUS register is accessible by a 16-bit read/write to address 'Base + 0x0002', but performing a 16-bit access to 'Base + 0x0003' is illegal.

#### 5.3.4.1 System Status Register (SSCM\_STATUS)

The System Status register is a read-only register that reflects the current state of the system.

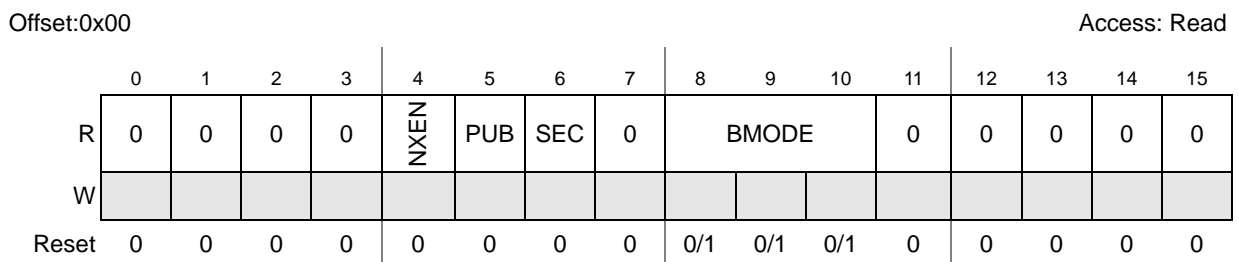


Figure 19. System Status Register (SSCM\_STATUS)

**Table 22. SSCM\_STATUS allowed register accesses**

Access type	8-bit	16-bit	32-bit <sup>(1)</sup>
Read	Allowed	Allowed	Allowed
Write	Not allowed	Not allowed	Not allowed

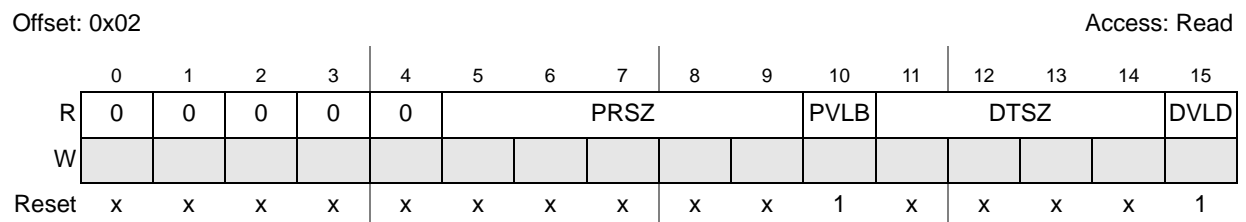
1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

**Table 23. SSCM\_STATUS field descriptions**

Field	Description
NXEN	Nexus enabled
PUB	Public Serial Access Status. This bit indicates whether serial boot mode with public password is allowed. 1 Serial boot mode with public password is allowed 0 Serial boot mode with private flash memory password is allowed
SEC	Security Status. This bit reflects the current security state of the flash memory. 1 The flash memory is secured. 0 The flash memory is not secured.
BMODE	Device Boot Mode 000 Reserved 001 FlexCAN_0 Serial Boot Loader 010 LINFlex_0 Serial Boot Loader 011 Single Chip 100 Reserved 101 Reserved 110 Reserved 111 Reserved This field is only updated during reset.

**5.3.4.2 System Memory Configuration Register (SSCM\_MEMCONFIG)**

The System Memory Configuration register is a read-only register that reflects the memory configuration of the system.



**Figure 20. System Memory Configuration Register (SSCM\_MEMCONFIG)**

**Table 24. SSCM\_MEMCONFIG field descriptions**

Field	Description
PRSZ	Code Flash Size 10000 128 KB 10001 256 KB 10010 384 KB 10011 512 KB 10101768 KB 10111 1 MB 11011 1.5 MB
PVLB	Code Flash Available This bit identifies whether or not the on-chip code Flash is available in the system memory map. The Flash may not be accessible due to security limitations, or because there is no Flash in the system. 1 Code Flash is available 0 Code Flash is not available
DTSZ	Data Flash Size 0000 No Data Flash 0011 64 KB
DVLD	Data Flash Valid This bit identifies whether or not the on-chip Data Flash is visible in the system memory map. The Flash may not be accessible due to security limitations, or because there is no Flash in the system. 1 Data Flash is visible 0 Data Flash is not visible

**Table 25. SSCM\_MEMCONFIG allowed register accesses**

Access type	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed (also reads SSCM_STATUS register)
Write	Not allowed	Not allowed	Not allowed

**5.3.4.3 Error Configuration (SSCM\_ERROR)**

The Error Configuration register is a read-write register that controls the error handling of the system.

Offset: 0x06

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAE	RAE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 21. Error Configuration (SSCM\_ERROR)**

**Table 26. SSCM\_ERROR field descriptions**

Field	Description
PAE	Peripheral Bus Abort Enable This bit enables bus aborts on any access to a peripheral slot that is not used on the device. This feature is intended to aid in debugging when developing application code. 1 Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception 0 Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception
RAE	Register Bus Abort Enable This bit enables bus aborts on illegal accesses to off-platform peripherals. Illegal accesses are defined as reads or writes to reserved addresses within the address space for a particular peripheral. This feature is intended to aid in debugging when developing application code. 1 Illegal accesses to peripherals produce a Prefetch or Data Abort exception 0 Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (that is, at the PBRIDGE level). In this case, bits PAE and RAE will have no effect on the abort.

**Table 27. SSCM\_ERROR allowed register accesses**

Access type	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed
Write	Allowed	Allowed	Not allowed

**5.3.4.4 Debug Status Port Register (SSCM\_DEBUGPORT)**

The Debug Status Port register is used to (optionally) provide debug data on a set of pins.

Offset: 0x08

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	DEBUG_MODE			
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure 22. Debug Status Port Register (SSCM\_DEBUGPORT)**

**Table 28. SSCM\_DEBUGPORT field descriptions**

Field	Description
DEBUG_MODE	Debug Status Port Mode This field selects the alternate debug functionality for the Debug Status Port. 000 No alternate functionality selected 001 Mode 1 selected 010 Mode 2 selected 011 Mode 3 selected 100 Mode 4 selected 101 Mode 5 selected 110 Mode 6 selected 111 Mode 7 selected <a href="#">Table 29</a> describes the functionality of the Debug Status Port in each mode.

**Table 29. Debug status port modes**

Pin (1)	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
0	SSCM_STATU S[0]	SSCM_STATU S[8]	SSCM_MEMCONFI G[0]	SSCM_MEMCONFI G[8]	Reserved	Reserved	Reserved
1	SSCM_STATU S[1]	SSCM_STATU S[9]	SSCM_MEMCONFI G[1]	SSCM_MEMCONFI G[9]	Reserved	Reserved	Reserved
2	SSCM_STATU S[2]	SSCM_STATU S[10]	SSCM_MEMCONFI G[2]	SSCM_MEMCONFI G[10]	Reserved	Reserved	Reserved
3	SSCM_STATU S[3]	SSCM_STATU S[11]	SSCM_MEMCONFI G[3]	SSCM_MEMCONFI G[11]	Reserved	Reserved	Reserved
4	SSCM_STATU S[4]	SSCM_STATU S[12]	SSCM_MEMCONFI G[4]	SSCM_MEMCONFI G[12]	Reserved	Reserved	Reserved
5	SSCM_STATU S[5]	SSCM_STATU S[13]	SSCM_MEMCONFI G[5]	SSCM_MEMCONFI G[13]	Reserved	Reserved	Reserved
6	SSCM_STATU S[6]	SSCM_STATU S[14]	SSCM_MEMCONFI G[6]	SSCM_MEMCONFI G[14]	Reserved	Reserved	Reserved
7	SSCM_STATU S[7]	SSCM_STATU S[15]	SSCM_MEMCONFI G[7]	SSCM_MEMCONFI G[15]	Reserved	Reserved	Reserved

1. All signals are active high, unless otherwise noted

PIN[0...7] referred to in [Table 29](#) equates to PC[2...9] (Pad 34...41).

**Table 30. SSCM\_DEBUGPORT allowed register accesses**

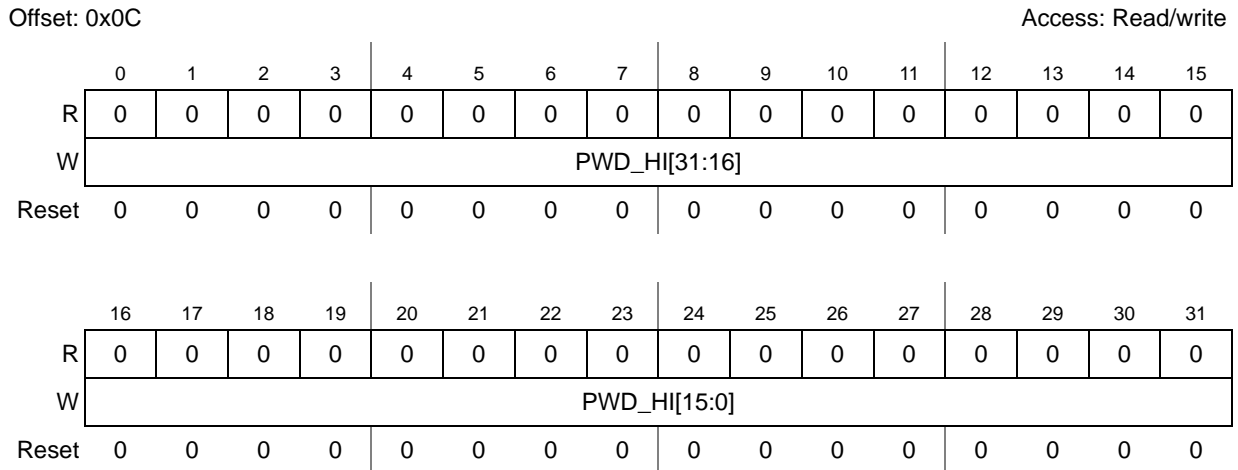
Access type	8-bit	16-bit	32-bit <sup>(1)</sup>
Read	Allowed	Allowed	Not allowed
Write	Allowed	Allowed	Not allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

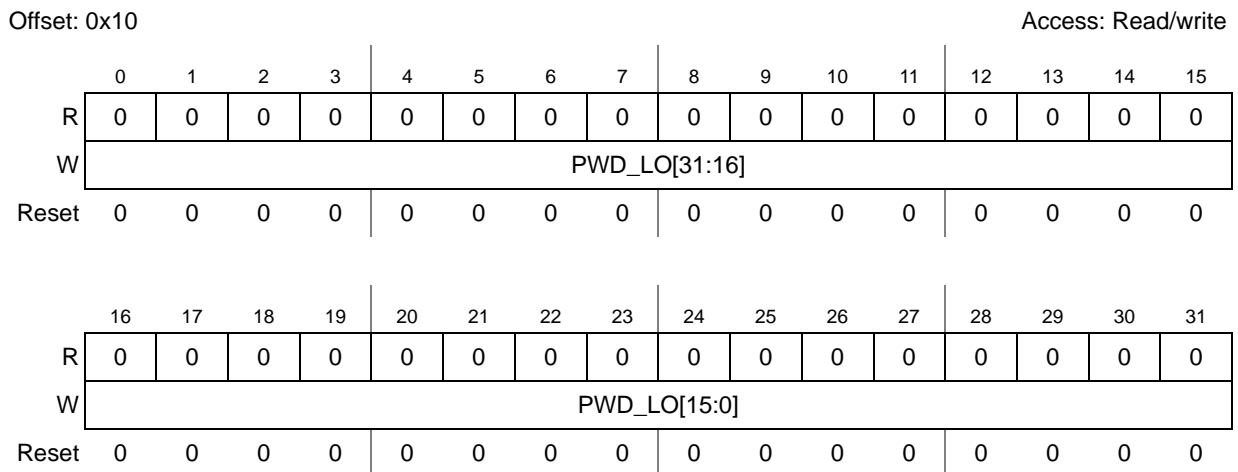
### 5.3.4.5 Password comparison registers

These registers provide a means for the BAM code to unsecure the device via the SSCM if the password has been provided via serial download.

**Figure 23. Password Comparison Register High Word (SSCM\_PWCMPH)**



**Figure 24. Password Comparison Register Low Word (SSCM\_PWC MPL)**



**Table 31. Password Comparison Register field descriptions**

Field	Description
PWD_HI	Upper 32 bits of the password
PWD_LO	Lower 32 bits of the password



Table 32. SSCM\_PWCMPH/L allowed register accesses

Access type	8-bit	16-bit	32-bit <sup>(1)</sup>
Read	Allowed	Allowed	Allowed
Write	Not allowed	Not allowed	Allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

In order to unsecure the device, the password needs to be written as follows: first the upper word to the SSCM\_PWCMPH register, then the lower word to the SSCM\_PWCMPPL register. The SSCM compares the 64-bit password entered into the SSCM\_PWCMPH / SSCM\_PWCMPPL registers with the NVPWM[1,0] private password stored in the shadow flash. If the passwords match then the SSCM temporarily uncensors the microcontroller.

## 6 Clock Description

This chapter describes the clock architectural implementation for SPC560B54/6x.

### 6.1 Clock architecture

System clocks are generated from three sources:

- Fast external crystal oscillator 4-16 MHz (FXOSC)
- Fast internal RC oscillator 16 MHz (FIRC)
- Frequency modulated phase locked loop (FMPLL)

Additionally, there are two low power oscillators:

- Slow internal RC oscillator 128 kHz (SIRC)
- Slow external crystal oscillator 32 KHz (SXOSC)

The clock architecture is shown in [Figure 25](#).

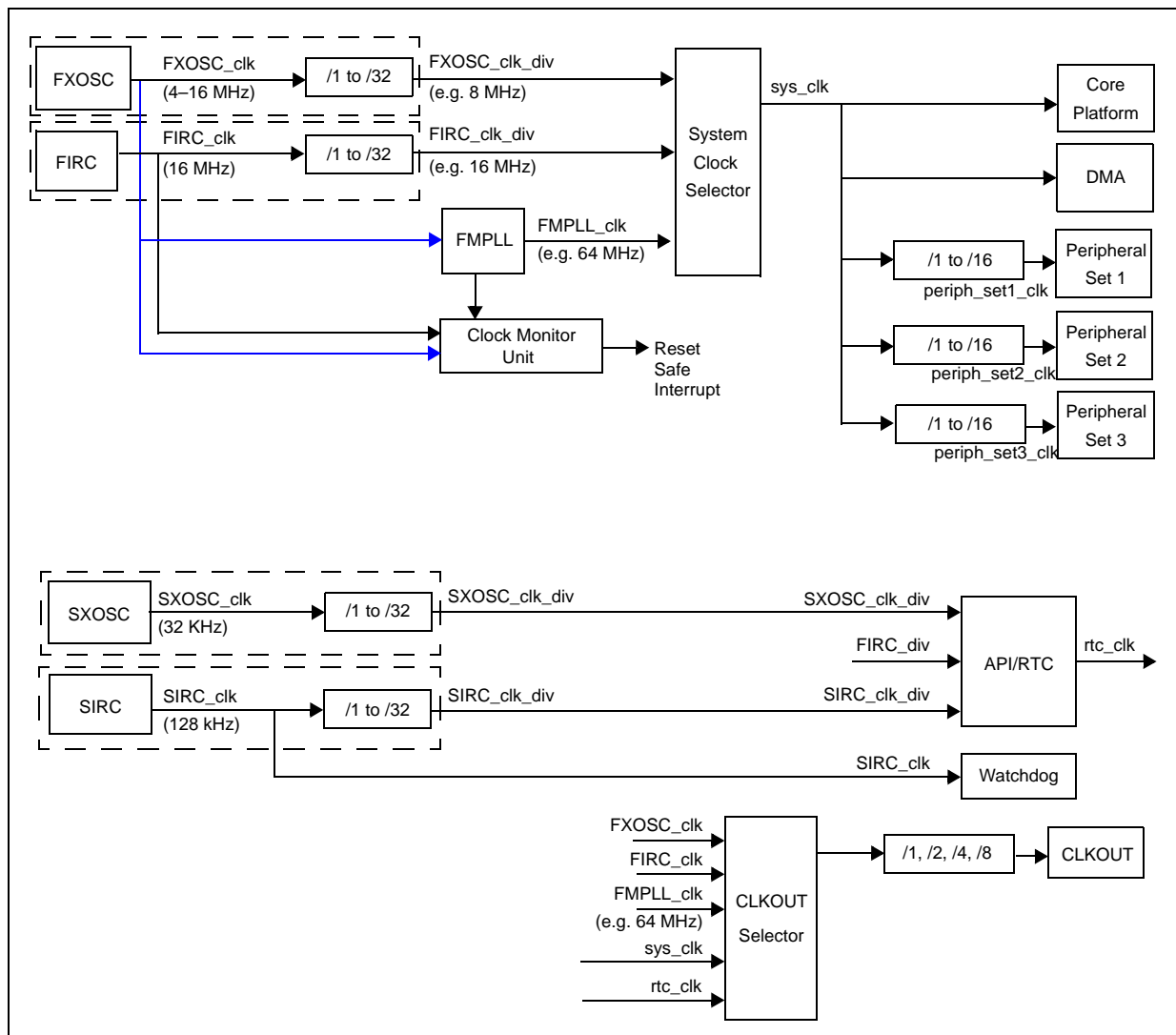


Figure 25. SPC560B54/6x system clock generation

## 6.2 Clock gating

The SPC560B54/6x provides the user with the possibility of gating the clock to the peripherals. [Table 33](#) describes for each peripheral the associated gating register address. See the ME\_PCTLn section in this reference manual.

Additionally, peripheral set (1, 2 or 3) frequency can be configured to be an integer (1 to 16) divided version of the main system clock. See the CGM\_SC\_DC0 section in this reference manual for details.

Table 33. SPC560B54/6x — Peripheral clock sources

Peripheral	Register gating address offset (base = 0xC3FDC0C0) <sup>(1)</sup>	Peripheral set <sup>(2)</sup>
RPP_ZOH Platform	none (managed through ME mode)	—
DSPI_n	4+n (n = 0..5)	2
FlexCAN_n	16+n (n = 0..5)	2
ADC_0	32	3
ADC_1	33	3
I <sup>2</sup> C	44	1
LINFLEX_n	48+n (n = 0..7)	1
LINFLEX_8	12	1
LINFLEX_9	13	1
CTU	57	3
CANS	60	—
SIUL	68	—
WKUP	69	—
eMIOS_n	72+n (n = 0..1)	3
RTC/API	91	—
PIT	92	—
CMU	104	—

1. See the ME\_PCTL section in this reference manual for details.

2. “—” means undivided system clock.

## 6.3 Fast external crystal oscillator (FXOSC) digital interface

The FXOSC digital interface controls the operation of the 4–16 MHz fast external crystal oscillator (FXOSC). It holds control and status registers accessible for application.

### 6.3.1 Main features

- Oscillator powerdown control and status reporting through MC\_ME block
- Oscillator clock available interrupt
- Oscillator bypass mode
- Output clock division factors ranging from 1, 2, 3....32

### 6.3.2 Functional description

The FXOSC circuit includes an internal oscillator driver and an external crystal circuitry. It provides an output clock that can be provided to the FMPLL or used as a reference clock to specific modules depending on system needs.

The FXOSC can be controlled by the MC\_ME module. The ME\_XXX\_MC[FXOSCON] bit controls the powerdown of the oscillator based on the current device mode while ME\_GS[S\_XOSC] register provides the oscillator clock available status.

After system reset, the oscillator is put into powerdown state and software has to switch on when required. Whenever the crystal oscillator is switched on from the off state, the OSCCNT counter starts and when it reaches the value EOCV[7:0]x512, the oscillator clock is made available to the system. Also, an interrupt pending FXOSC\_CTL[I\_OSC] bit is set. An interrupt is generated if the interrupt mask bit M\_OSC is set.

The oscillator circuit can be bypassed by setting FXOSC\_CTL[OSCBYP]. This bit can only be set by software. A system reset is needed to reset this bit. In this bypass mode, the output clock has the same polarity as the external clock applied on the EXTAL pin and the oscillator status is forced to '1'. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 34 shows the truth table of different oscillator configurations.

**Table 34. Truth table of crystal oscillator**

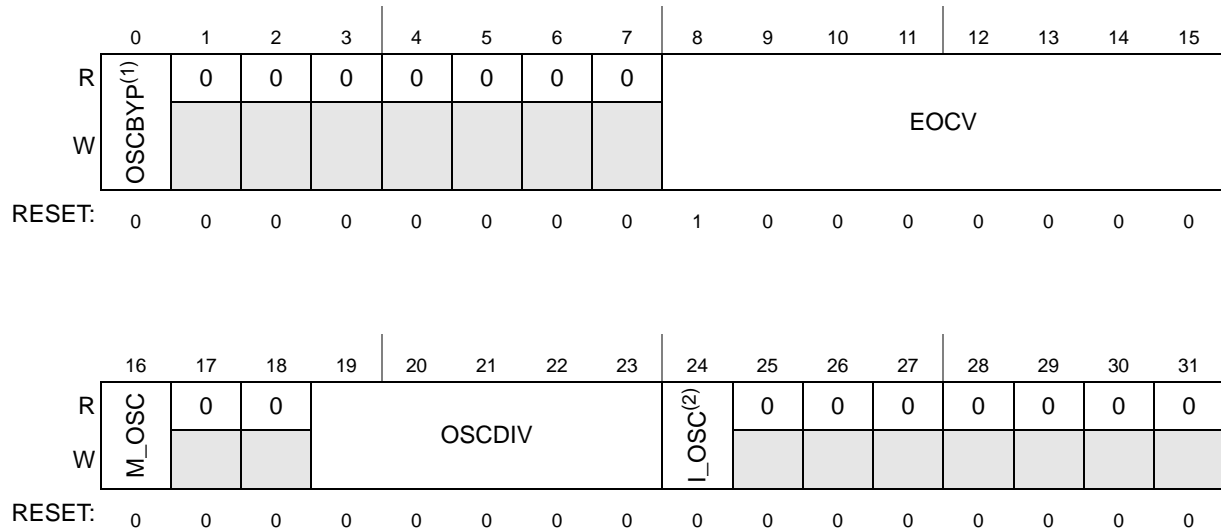
ME_XXX_MC[FXOSCON]	FXOSC_CTL[OSCBYP]	XTAL	EXTAL	FXOSC	Oscillator mode
0	0	No crystal, High Z	No crystal, High Z	0	Powerdown, IDDQ
x	1	x	Ext clock	EXTAL	Bypass, OSC disabled
1	0	Crystal	Crystal	EXTAL	Normal, OSC enabled
		Gnd	Ext clock	EXTAL	Normal, OSC enabled

The FXOSC clock can be further divided by a configurable factor in the range 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by FXOSC\_CTL[OSCDIV] field.

### 6.3.3 Register description

Address: 0xC3FE\_0000

Access: Special read/write



**Figure 26. Fast External Crystal Oscillator Control Register (FXOSC\_CTL)**

1. You can read this field, and you can write a value of "1" to it. Writing a "0" has no effect. A reset will also clear this bit.
2. You can write a value of "0" or "1" to this field. However, writing a "1" will clear this field, and writing "0" will have no effect on the field value.

**Table 35. FXOSC\_CTL field descriptions**

Field	Description
OSCBYP	Crystal Oscillator bypass. This bit specifies whether the oscillator should be bypassed or not. 0 Oscillator output is used as root clock 1 EXTAL is used as root clock
EOCV	End of Count Value. These bits specify the end of count value to be used for comparison by the oscillator stabilization counter OSCCNT after reset or whenever it is switched on from the off state (OSCCNT runs on the FXOSC). This counting period ensures that external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV × 512, the crystal oscillator clock interrupt (I_OSC) request is generated. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected.
M_OSC	Crystal oscillator clock interrupt mask. 0 Crystal oscillator clock interrupt is masked. 1 Crystal oscillator clock interrupt is enabled.
OSCDIV	Crystal oscillator clock division factor. This field specifies the crystal oscillator output clock division factor. The output clock is divided by the factor OSCDIV+1.
I_OSC	Crystal oscillator clock interrupt. This bit is set by hardware when OSCCNT counter reaches the count value EOCV × 512. 0 No oscillator clock interrupt occurred. 1 Oscillator clock interrupt pending.

## 6.4 Slow external crystal oscillator (SXOSC) digital interface

### 6.4.1 Introduction

The SXOSC digital interface controls the operation of the 32 KHz slow external crystal oscillator (SXOSC). It holds control and status registers accessible for application.

### 6.4.2 Main features

- Oscillator powerdown control and status
- Oscillator clock available interrupt
- Oscillator bypass mode
- Output clock division factors ranging from 1 to 32

### 6.4.3 Functional description

The SXOSC circuit includes an internal oscillator driver and an external crystal circuitry. It can be used as a reference clock to specific modules depending on system needs.

The SXOSC can be controlled via the SXOSC\_CTL register. The OSCON bit controls the powerdown while bit S\_OSC provides the oscillator clock available status.

After system reset, the oscillator is put to powerdown state and software has to switch on when required. Whenever the SXOSC is switched on from off state, the OSCCNT counter starts and when it reaches the value EOCV[7:0]×512, the oscillator clock is made available to the system. Also, an interrupt pending SXOSC\_CTL[I\_OSC] bit is set. An interrupt will be generated if the interrupt mask bit M\_OSC is set.

The oscillator circuit can be bypassed by writing SXOSC\_CTL[OSCBYP] bit to '1'. This bit can only be set by software. A system reset is needed to reset this bit. In this bypass mode, the output clock has the same polarity as the external clock applied on the OSC32K\_EXTAL pin and the oscillator status is forced to '1'. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 36 shows the truth table of different configurations of the oscillator.

**Table 36. SXOSC truth table**

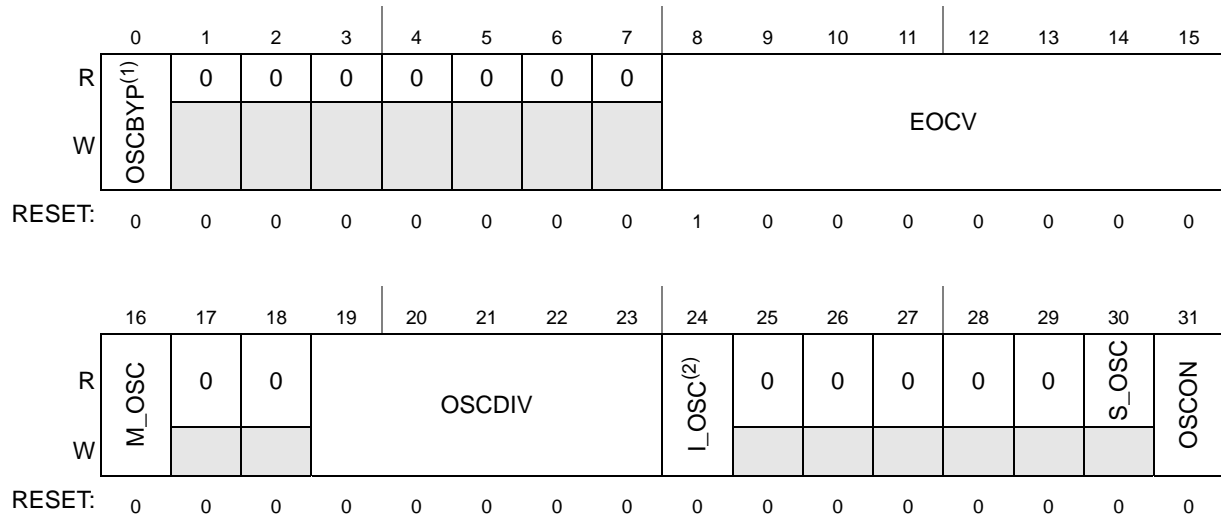
SXOSC_CTL fields		OSC32K_XTAL	OSC32K_EXTAL	SXOSC	Oscillator MODE
OSCON	OSCBYP				
0	0	No crystal, High Z	No crystal, High Z	0	Powerdown, IDDQ
x	1	x	External clock	OSC32K_EXTAL	Bypass, OSC disabled
1	0	Crystal	Crystal	OSC32K_EXTAL	Normal, OSC enabled
		Ground	External clock	OSC32K_EXTAL	Normal, OSC enabled

The SXOSC clock can be further divided by a configurable factor in the range 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by SXOSC\_CTL[OSCDIV] field.

### 6.4.4 Register description

Address: 0xC3FE\_0040

Access: Special read/write



**Figure 27. Slow External Crystal Oscillator Control Register (SXOSC\_CTL)**

1. You can read this field, and you can write a value of "1" to it. Writing a "0" has no effect. A reset will also clear this bit.
2. You can write a value of "0" or "1" to this field. However, writing a "1" will clear this field, and writing "0" will have no effect on the field value.

**Table 37. SXOSC\_CTL field descriptions**

Field	Description
OSCIBYP	Crystal Oscillator bypass. This bit specifies whether the oscillator should be bypassed or not. 0 Oscillator output is used as root clock. 1 OSC32K_EXTAL is used as root clock.
EOCV	End of Count Value. This field specifies the end of count value to be used for comparison by the oscillator stabilization counter OSCCNT after reset or whenever it is switched on from the off state. This counting period ensures that external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV × 512, the crystal oscillator clock interrupt (I_OSC) request is generated. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected.
M_OSC	Crystal oscillator clock interrupt mask. 0 Crystal oscillator clock interrupt is masked. 1 Crystal oscillator clock interrupt is enabled.
OSCDIV	Crystal oscillator clock division factor. This field specifies the crystal oscillator output clock division factor. The output clock is divided by the factor OSCDIV + 1.
I_OSC	Crystal oscillator clock interrupt. This field is set by hardware when OSCCNT counter reaches the count value EOCV × 512. 0 No oscillator clock interrupt occurred. 1 Oscillator clock interrupt pending.





Table 37. SXOSC\_CTL field descriptions (continued)

Field	Description
S_OSC	Crystal oscillator status. 0 Crystal oscillator output clock is not stable. 1 Crystal oscillator is providing a stable clock.
OSCON	Crystal oscillator enable. 0 Crystal oscillator is switched off. 1 Crystal oscillator is switched on.

*Note:* The 32 KHz slow external crystal oscillator is by default always ON, but can be configured OFF in standby by setting the OSCON bit.

## 6.5 Slow internal RC oscillator (SIRC) digital interface

### 6.5.1 Introduction

The SIRC digital interface controls the 128 kHz slow internal RC oscillator (SIRC). It holds control and status registers accessible for application.

### 6.5.2 Functional description

The SIRC provides a low frequency ( $f_{SIRC}$ ) clock of 128 kHz requiring very low current consumption. This clock can be used as the reference clock when a fixed base time is required for specific modules.

SIRC is always on in all device modes except STANDBY mode. In STANDBY mode, it is controlled by SIRC\_CTL[SIRCON\_STDBY] bit. The clock source status is updated in SIRC\_CTL[S\_SIRC] bit.

The SIRC clock can be further divided by a configurable division factor in the range from 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by SIRC\_CTL[SIRCDIV] bits.

The SIRC output frequency can be trimmed using SIRC\_CTL[SIRCTRIM]. After a power-on reset, the SIRC is trimmed using a factory test value stored in test flash memory. However, after a power-on reset the test flash memory value is not visible at SIRC\_CTL[SIRCTRIM] and this field shows a value of zero. Therefore, be aware that the SIRC\_CTL[SIRCTRIM] does not reflect the current trim value until you have written to this field. Pay particular attention to this feature when you initiate a read-modify-write operation on SIRC\_CTL, because a SIRCTRIM value of zero may be unintentionally written back and this may alter the SIRC frequency. In this case, you should calibrate the SIRC using the CMU or be sure that you only write to the upper 16 bits of this SIRC\_CTL.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from -16 to 15. As the trimming code increases, the internal time constant increases and frequency reduces. Please refer to device datasheet for average frequency variation of the trimming step.

### 6.5.3 Register description

Address: 0xC3FE\_0080

Access: Read/write

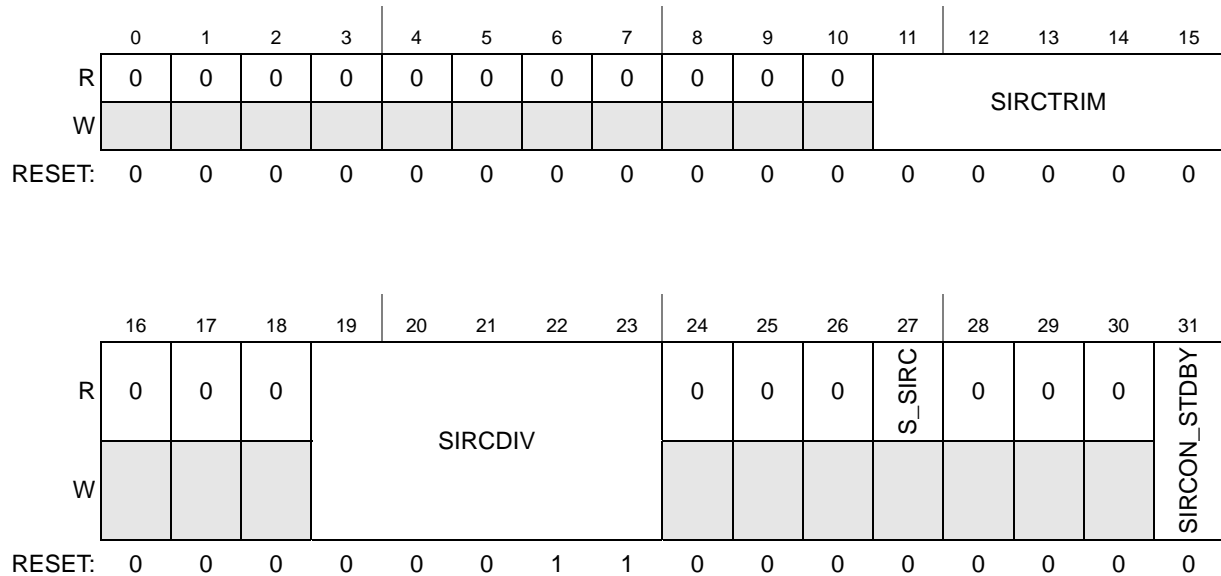


Figure 28. Low Power RC Control Register (SIRC\_CTL)

Table 38. SIRC\_CTL field descriptions

Field	Description
SIRCTRIM	SIRC trimming bits. This field corresponds (via two's complement) to a trim factor of -16 to +15. A +1 change in SIRCTRIM decreases the current frequency by $\Delta_{SIRCTRIM}$ (see the device data sheet). A -1 change in SIRCTRIM increases the current frequency by $\Delta_{SIRCTRIM}$ (see the device data sheet).
SIRCDIV	SIRC clock division factor. This field specifies the SIRC oscillator output clock division factor. The output clock is divided by the factor SIRCDIV+1.
S_SIRC	SIRC clock status. 0 SIRC is not providing a stable clock. 1 SIRC is providing a stable clock.
SIRCON_STDBY	SIRC control in STANDBY mode. 0 SIRC is switched off in STANDBY mode. 1 SIRC is switched on in STANDBY mode.

## 6.6 Fast internal RC oscillator (FIRC) digital interface

### 6.6.1 Introduction

The FIRC digital interface controls the 16 MHz fast internal RC oscillator (FIRC). It holds control and status registers accessible for application.

## 6.6.2 Functional description

The FIRC provides a high frequency ( $f_{\text{FIRC}}$ ) clock of 16 MHz. This clock can be used to accelerate the exit from reset and wakeup sequence from low power modes of the system. It is controlled by the MC\_ME module based on the current device mode. The clock source status is updated in ME\_GS[S\_RC]. Please refer to the MC\_ME chapter for further details.

The FIRC can be further divided by a configurable division factor in the range from 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by RC\_CTL[RCDIV] bits.

The FIRC output frequency can be trimmed using FIRC\_CTL[FIRCTRIM]. After a power-on reset, the FIRC is trimmed using a factory test value stored in test flash memory. However, after a power-on reset the test flash memory value is not visible at FIRC\_CTL[FIRCTRIM], and this field will show a value of zero. Therefore, be aware that the FIRC\_CTL[FIRCTRIM] field does not reflect the current trim value until you have written to it. Pay particular attention to this feature when you initiate a read-modify-write operation on FIRC\_CTL, because a FIRCTRIM value of zero may be unintentionally written back and this may alter the FIRC frequency. In this case, you should calibrate the FIRC using the CMU or ensure that you write only to the upper 16 bits of this FIRC\_CTL.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from -32 to 31. As the trimming code increases, the internal time constant increases and frequency reduces. Please refer to device datasheet for average frequency variation of the trimming step.

During STANDBY mode entry process, the FIRC is controlled based on ME\_STANDBY\_MC[RCON] bit. This is the last step in the standby entry sequence. On any system wake-up event, the device exits STANDBY mode and switches on the FIRC. The actual powerdown status of the FIRC when the device is in standby is provided by RC\_CTL[FIRCON\_STDBY] bit.

### 6.6.3 Register description

Address: 0xC3FE\_0060

Access: Read/write

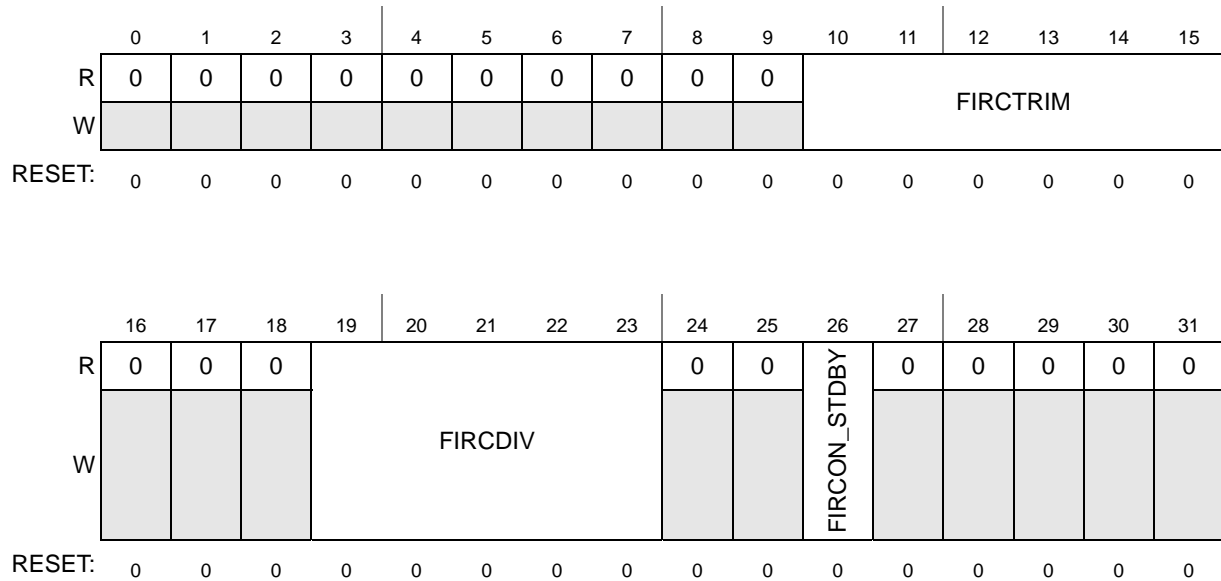


Figure 29. FIRC Oscillator Control Register (FIRC\_CTL)

Table 39. FIRC\_CTL field descriptions

Field	Description
FIRCTRIM	FIRC trimming bits. This field corresponds (via two's complement) to a trim factor of -16 to +15. A +1 change in FIRCTRIM decreases the current frequency by $\Delta_{FIRCTRIM}$ (see the device data sheet). A -1 change in SIRCTRIM increases the current frequency by $\Delta_{FIRCTRIM}$ (see the device data sheet).
FIRCDIV	FIRC clock division factor. This field specifies the FIRC oscillator output clock division factor. The output clock is divided by the factor FIRCDIV+1.
FIRCON_STDBY	FIRC control in STANDBY mode. 0 FIRC is switched off in STANDBY mode. 1 FIRC is in STANDBY mode.

## 6.7 Frequency-modulated phase-locked loop (FMPLL)

### 6.7.1 Introduction

This section describes the features and functions of the FMPLL module implemented in the device.

## 6.7.2 Overview

The FMPLL enables the generation of high speed system clocks from a common 4–16 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor and output clock divider ratio are all software configurable.

SPC560B54/6x has one FMPLL that can generate the system clock and takes advantage of the FM mode.

*Note:* The user must take care not to program device with a frequency higher than allowed (no hardware check).

The FMPLL block diagram is shown in [Figure 30](#).

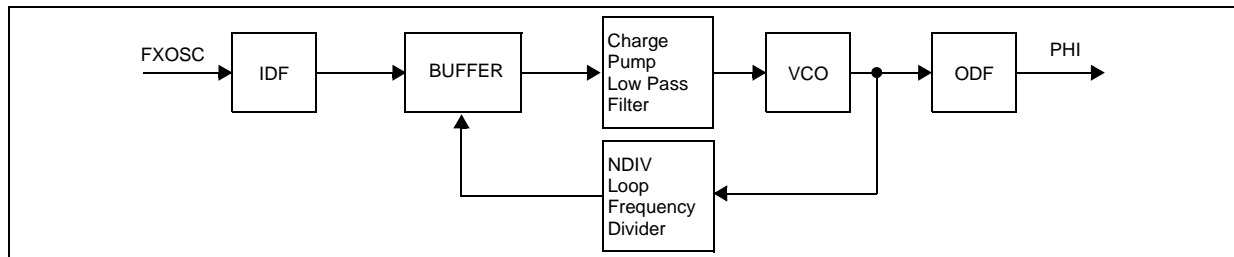


Figure 30. FMPLL block diagram

## 6.7.3 Features

The FMPLL has the following major features:

- Input clock frequency 4 MHz – 16 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- Frequency divider (FD) for reduced frequency operation without forcing the FMPLL to relock
- Frequency modulated FMPLL
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth
  - $\pm 0.25\%$  to  $\pm 4\%$  deviation from center spread frequency<sup>(g)</sup>
  - $-0.5\%$  to  $+8\%$  deviation from down spread frequency
  - Programmable modulation frequency dependent on reference frequency
- Self-clocked mode (SCM) operation
- 4 available modes
  - Normal mode
  - Progressive clock switching
  - Normal mode with frequency modulation
  - Powerdown mode

g. Spread spectrum should be programmed in line with maximum datasheet frequency figures.

### 6.7.4 Memory map<sup>(h)</sup>

Table 40 shows the memory map of the FMPLL.

**Table 40. FMPLL memory map**

Base address: 0xC3FE_00A0		
Address offset	Register	Location
0x0	Control Register (CR)	on page 150
0x4	Modulation Register (MR)	on page 152

### 6.7.5 Register description

The FMPLL operation is controlled by two registers. Those registers can be accessed and written in supervisor mode only.

#### 6.7.5.1 Control Register (CR)

Offset: 0x0 Access: Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	IDF				ODF		0	NDIV						
W																
Reset	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	EN_PLL_SW	0	UNLOCK_ONCE	0	I_LOCK	S_LOCK	PLL_FAIL_MASK	PLL_FAIL_FLAG	1
W												w1c		PLL_FAIL_MASK	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

**Figure 31. Control Register (CR)**

**Table 41. CR field descriptions**

Field	Description
IDF	The value of this field sets the FMPLL input division factor as described in Table 42.
ODF	The value of this field sets the FMPLL output division factor as described in Table 43.
NDIV	The value of this field sets the FMPLL loop division factor as described in Table 44.

h. FMPLL\_x are mapped through the ME\_CGM register slot.

**Table 41. CR field descriptions (continued)**

Field	Description
EN_PLL_SW	This bit is used to enable progressive clock switching. After the PLL locks, the PLL output initially is divided by 8, and then progressively decreases until it reaches divide-by-1. 0 Progressive clock switching disabled. 1 Progressive clock switching enabled. Note: Progressive clock switching should not be used if a non-changing clock is needed, such as for serial communications, until the division has finished.
UNLOCK_ONCE	This bit is a sticking indication of FMPLL loss of lock condition. UNLOCK_ONCE is set when the FMPLL loses lock. Whenever the FMPLL reacquires lock, UNLOCK_ONCE remains set. Only a power-on reset clears this bit.
I_LOCK	This bit is set by hardware whenever there is a lock/unlock event.
S_LOCK	This bit is an indication of whether the FMPLL has acquired lock. 0: FMPLL unlocked 1: FMPLL locked
PLL_FAIL_MASK	This bit is used to mask the pll_fail output. 0 pll_fail not masked. 1 pll_fail masked.
PLL_FAIL_FLAG	This bit is asynchronously set by hardware whenever a loss of lock event occurs while FMPLL is switched on. It is cleared by software writing '1'.

**Table 42. Input divide ratios**

IDF[3:0]	Input divide ratios
0000	Divide by 1
0001	Divide by 2
0010	Divide by 3
0011	Divide by 4
0100	Divide by 5
0101	Divide by 6
0110	Divide by 7
0111	Divide by 8
1000	Divide by 9
1001	Divide by 10
1010	Divide by 11
1011	Divide by 12
1100	Divide by 13
1101	Divide by 14
1110	Divide by 15
1111	Clock Inhibit

Table 43. Output divide ratios

ODF[1:0]	Output divide ratios
00	Divide by 2
01	Divide by 4
10	Divide by 8
11	Divide by 16

Table 44. Loop divide ratios

NDIV[6:0]	Loop divide ratios
0000000–0011111	—
0100000	Divide by 32
0100001	Divide by 33
0100010	Divide by 34
...	...
1011111	Divide by 95
1100000	Divide by 96
1100001–1111111	—

6.7.5.2 Modulation Register (MR)

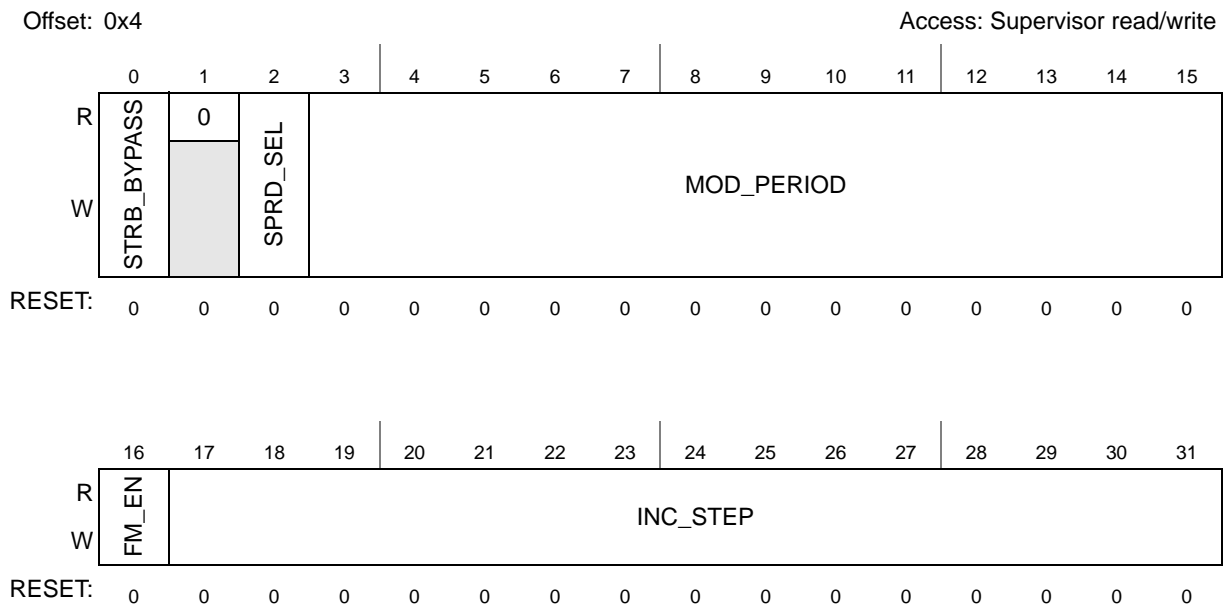


Figure 32. Modulation Register (MR)



**Table 45. MR field descriptions**

Field	Description
STRB_BYPASS	Strobe bypass. The STRB_BYPASS signal is used to bypass the strobe signal used inside FMPLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD and SPRD_SEL). 0 Strobe is used to latch FMPLL modulation control bits 1 Strobe is bypassed. In this case control bits need to be static. The control bits must be changed only when FMPLL is in powerdown mode.
SPRD_SEL	Spread type selection. The SPRD_SEL controls the spread type in Frequency Modulation mode. 0 Center SPREAD 1 Down SPREAD
MOD_PERIOD	Modulation period. The MOD_PERIOD field is the binary equivalent of the value modperiod derived from following formula: $\text{modperiod} = \frac{f_{ref}}{4 \times f_{mod}}$ where: $f_{ref}$ : represents the frequency of the feedback divider $f_{mod}$ : represents the modulation frequency
FM_EN	Frequency Modulation Enable. The FM_EN enables the frequency modulation. 0 Frequency modulation disabled 1 Frequency modulation enabled
INC_STEP	Increment step. The INC_STEP field is the binary equivalent of the value incstep derived from following formula: $\text{incstep} = \text{round}\left(\frac{(2^{15} - 1) \times \text{md} \times \text{MDF}}{100 \times 5 \times \text{MODPERIOD}}\right)$ where: md: represents the peak modulation depth in percentage (Center spread -- pk-pk=+/-md, Downspread -- pk-pk=-2×md) MDF: represents the nominal value of loop divider (CR[NDIV])

**6.7.6 Functional description**

**6.7.6.1 Normal mode**

In Normal Mode the FMPLL inputs are driven by the CR. This means that, when the FMPLL is in lock state, the FMPLL output clock (PHI) is derived by the reference clock (CLKIN) through this relation:

$$\text{phi} = \frac{\text{clkin} \cdot \text{NDIV}}{\text{IDF} \cdot \text{ODF}}$$

where the value of IDF, NDIV and ODF are set in the CR and can be derived from [Table 42](#), [Table 43](#) and [Table 44](#).

**Table 46. FMPLL lookup table**

Crystal frequency (MHz)	FMPLL output frequency (MHz)	CR field values			VCO frequency (MHz)
		IDF	ODF	NDIV	
8	32	0	2	32	256
	64	0	2	64	512
	80	0	1	40	320
16	32	1	2	32	256
	64	1	2	64	512
	80	1	1	40	320
40	32	4	2	32	256
	64	4	2	64	512
	80	3	1	32	320

**6.7.6.2 Progressive clock switching**

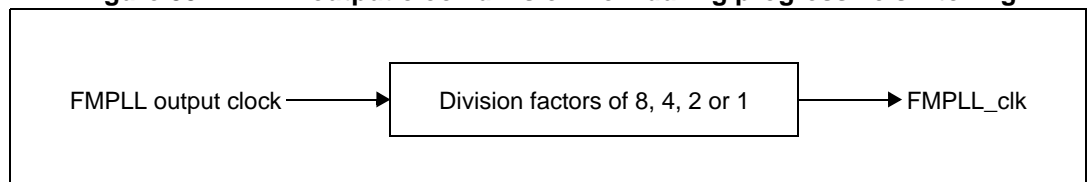
Progressive clock switching allows to switch the system clock to FMPLL output clock stepping through different division factors. This means that the current consumption gradually increases and, in turn, voltage regulator response is improved.

This feature can be enabled by programming CR[EN\_PLL\_SW] bit. When enabled, the system clock is switched to divided PHI. The FMPLL\_clk divider is then progressively decreased to the target divider as shown in [Table 47](#).

**Table 47. Progressive clock switching on pll\_select rising edge**

Number of FMPLL output clock cycles	FMPLL_clk frequency (FMPLL output clock frequency)
8	(FMPLL output clock frequency)/8
16	(FMPLL output clock frequency)/4
32	(FMPLL output clock frequency)/2
onward	FMPLL output clock frequency

**Figure 33. FMPLL output clock division flow during progressive switching**



### 6.7.6.3 Normal mode with frequency modulation

The FMPLL default mode is without frequency modulation enabled. When frequency modulation is enabled, however, two parameters must be set to generate the desired level of modulation: the PERIOD, and the STEP. The modulation waveform is always a triangle wave and its shape is not programmable.

FM mode is activated in two steps:

1. Configure the FM mode characteristics: MOD\_PERIOD, INC\_STEP.
2. Enable the FM mode by programming bit FM\_EN of the MR to '1'. FM mode can only be enabled when FMPLL is in lock state.

There are two ways to latch these values inside the FMPLL, depending on the value of bit STRB\_BYPASS in the MR.

If STRB\_BYPASS is low, the modulation parameters are latched in the FMPLL only when the strobe signal goes high for at least two cycles of CLKIN clock. The strobe signal is automatically generated in the FMPLL digital interface when the modulation is enabled (FM\_EN goes high) if the FMPLL is locked (S\_LOCK = 1) or when the modulation has been enabled (FM\_EN = 1) and FMPLL enters lock state (S\_LOCK goes high).

If STRB\_BYPASS is high, the strobe signal is bypassed. In this case, control bits (MOD\_PERIOD[12:0], INC\_STEP[14:0], SPREAD\_CONTROL) need to be static or hardwired to constant values. The control bits must be changed only when the FMPLL is in powerdown mode.

The modulation depth in % is

$$\text{ModulationDepth} = \left( \frac{100 \times 5 \times \text{INCSTEP} \times \text{MODPERIOD}}{(2^{15} - 1) \times \text{MDF}} \right)$$

*Note:* The user must ensure that the product of INCSTEP and MODPERIOD is less than  $(2^{15}-1)$ .

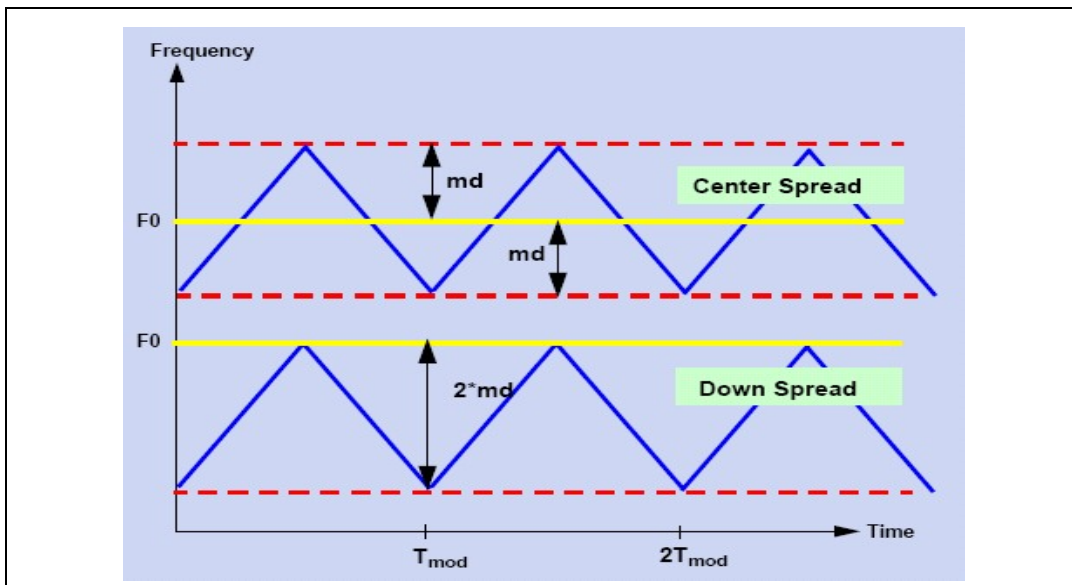


Figure 34. Frequency modulation

#### 6.7.6.4 Powerdown mode

To reduce consumption, the FMPLL can be switched off when not required by programming the registers ME\_x\_MC on the MC\_ME module.

#### 6.7.7 Recommendations

To avoid any unpredictable behavior of the FMPLL clock, it is recommended to follow these guidelines:

- The FMPLL VCO frequency should reside in the range 256 MHz to 512 MHz. Care is required when programming the multiplication and division factors to respect this requirement.
- Once the PLL has been locked, only the output divider can be changed.
- Use progressive clock switching if system clock changes are required while the PLL is being used as the system clock source. MOD\_PERIOD, INC\_STEP, SPREAD\_SEL bits should be modified before activating the FM mode. Then strobe has to be generated to enable the new settings. If STRB\_BYP is set to 1 then MOD\_PERIOD, INC\_STEP, and SPREAD\_SEL can be modified only when FMPLL is in powerdown mode.
- Use progressive clock switching (FMPLL output clock can be changed when it is the system clock, but only when using progressive clock switching).

### 6.8 Clock monitor unit (CMU)

#### 6.8.1 Introduction

The Clock Monitor Unit (CMU), also referred to as Clock Quality Checker or Clock Fault Detector, serves two purposes. The main task is to permanently supervise the integrity of the various clock sources, for example a crystal oscillator or FMPLL. In case the FMPLL leaves an upper or lower frequency boundary or the crystal oscillator fails it can detect and forward these kind of events towards the MC\_ME and MC\_CGM. The clock management unit in turn can then switch to a SAFE mode where it uses the default safe clock source (FIRC), reset the device or generate the interrupt according to the system needs.

It can also monitor the external crystal oscillator clock, which must be greater than the internal RC clock divided by a division factor given by CMU\_CSR[RCDIV], and generates a system clock transition request or an interrupt when enabled.

The second task of the CMU is to provide a frequency meter, which allows to measure the frequency of one clock source vs. a reference clock. This is useful to allow the calibration of the on-chip RC oscillator(s), as well as being able to correct/calculate the time deviation of a counter which is clocked by the RC oscillator.

#### 6.8.2 Main features

- FIRC, SIRC, SXOSC oscillator frequency measurement using FXOSC as reference clock
- External oscillator clock monitoring with respect to FIRC\_clk/n clock
- FMPLL clock frequency monitoring for a high and low frequency range with FIRC as reference clock
- Event generation for various failures detected inside monitoring unit

### 6.8.3 Block diagram

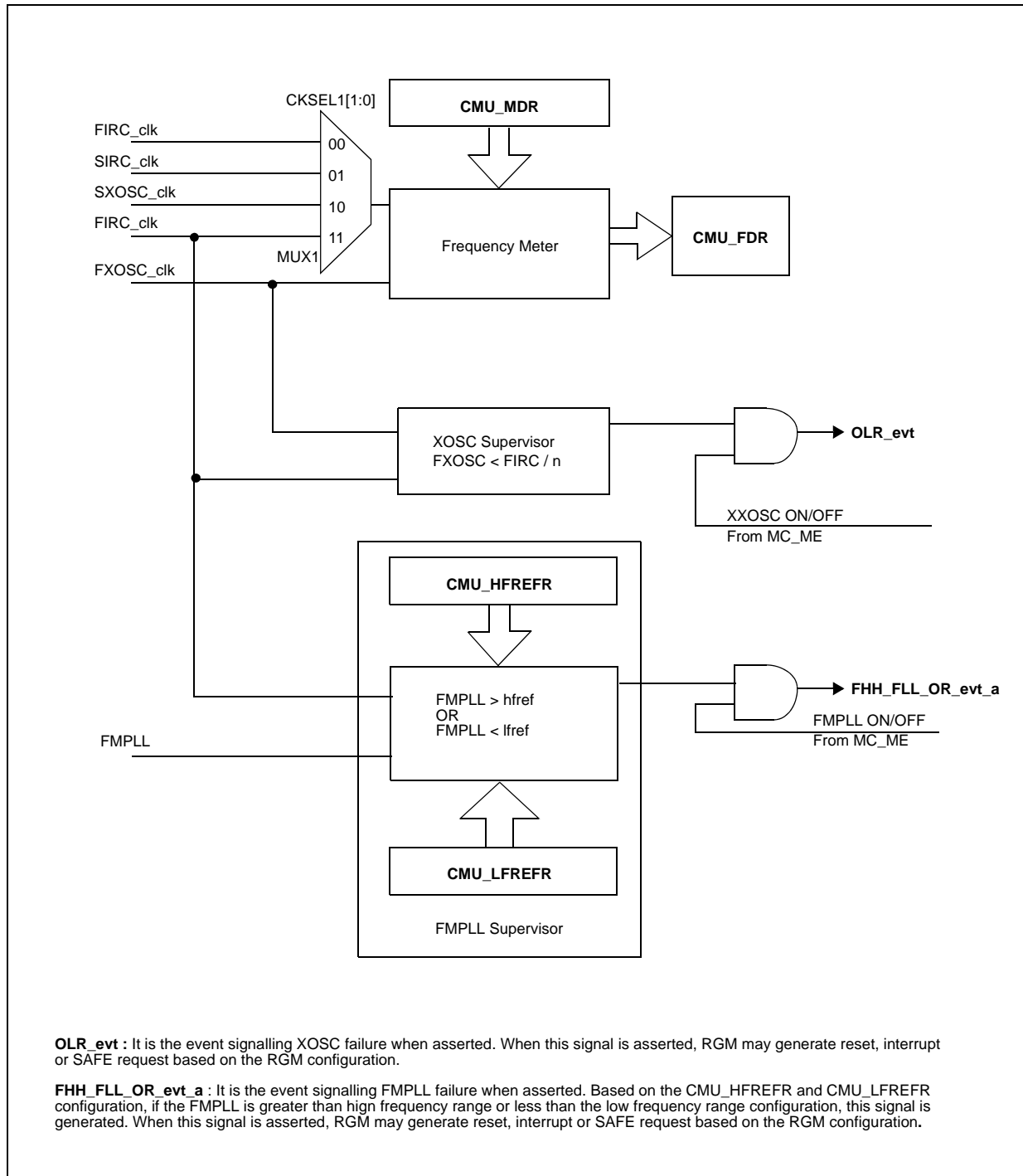


Figure 35. Clock Monitor Unit diagram

## 6.8.4 Functional description

The clock and frequency names referenced in this block are defined as follows:

- FXOSC\_clk: clock coming from the fast external crystal oscillator
- SXOSC\_clk: clock coming from the slow external crystal oscillator
- SIRC\_clk: clock coming from the slow (low frequency) internal RC oscillator
- FIRC\_clk: clock coming from the fast (high frequency) internal RC oscillator
- FMPLL\_clk: clock coming from the FMPLL
- $f_{\text{FXOSC\_clk}}$ : frequency of fast external crystal oscillator clock
- $f_{\text{SXOSC\_clk}}$ : frequency of slow external crystal oscillator clock
- $f_{\text{SIRC\_clk}}$ : frequency of slow (low frequency) internal RC oscillator
- $f_{\text{FIRC\_clk}}$ : frequency of fast (high frequency) internal RC oscillator
- $f_{\text{FMPLL\_clk}}$ : frequency of FMPLL clock

### 6.8.4.1 Crystal clock monitor

If  $f_{\text{FXOSC\_clk}}$  is less than  $f_{\text{FIRC\_clk}}$  divided by  $2^{\text{RCDIV}}$  bits of the CMU\_CSR and the FXOSC\_clk is 'ON' as signalled by the MC\_ME then:

- An event pending bit OLRI in CMU\_ISR is set.
- A failure event OLR is signalled to the MC\_RGM which in turn can automatically switch to a safe fallback clock and generate an interrupt or reset.

*Note:* Functional FXOSC monitoring can only be guaranteed when the FXOSC frequency is greater than  $(\text{FIRC} / 2^{\text{RCDIV}}) + 0.5 \text{ MHz}$ .

### 6.8.4.2 FMPLL clock monitor

The  $f_{\text{FMPLL\_clk}}$  can be monitored by programming bit CME of the CMU\_CSR register to '1'. The FMPLL\_clk monitor starts as soon as bit CME is set. This monitor can be disabled at any time by writing bit CME to '0'.

If  $f_{\text{FMPLL\_clk}}$  is greater than a reference value determined by bits HFREF[11:0] of the CMU\_HFREFR and the FMPLL\_clk is 'ON', as signalled by the MC\_ME, then:

- An event pending bit FHHI in CMU\_ISR is set.
- A failure event is signalled to the MC\_RGM which in turn can generate an interrupt or safe mode request or functional reset depending on the programming model.

If  $f_{\text{FMPLL\_clk}}$  is less than a reference value determined by bits LFREF[11:0] of the CMU\_LFREFR and the FMPLL\_clk is 'ON', as signalled by the MC\_ME, then:

- An event pending bit FLLI in CMU\_ISR is set.
- A failure event FLL is signalled to the MC\_RGM which in turn can generate an interrupt or safe mode request or functional reset depending on the programming model.

*Note:* The internal RC oscillator is used as reliable reference clock for the clock supervision. In order to avoid false events, proper programming of the dividers is required. These have to take into account the accuracy and frequency deviation of the internal RC oscillator.

*Note:* If PLL frequency goes out of range, the CMU shall generate FMPLL fill/fhh event. It takes approximately 5  $\mu\text{s}$  to generate this event.

*Note:* Functional FMPLL monitoring can only be guaranteed when the FMPLL frequency is greater than  $(\text{FIRC} / 4) + 0.5 \text{ MHz}$ .

### 6.8.4.3 Frequency meter

The purpose of the frequency meter is twofold:

- to measure the frequency of the oscillators SIRC, FIRC or SXOSC
- to calibrate an internal RC oscillator (SIRC or FIRC) using a known frequency

**Hint:** This value can then be stored into the flash so that application software can reuse it later on.

The reference clock is always the FXOSC\_clk. The frequency meter returns a precise value of frequencies  $f_{SXOSC\_clk}$ ,  $f_{FIRC\_clk}$  or  $f_{SIRC\_clk}$  according to CKSEL1 bit value. The measure starts when bit SFM (Start Frequency Measure) in the CMU\_CSR is set to '1'. The measurement duration is given by the CMU\_MDR in numbers of clock cycles of the selected clock source with a width of 20 bits. Bit SFM is reset to '0' by hardware once the frequency measurement is done and the count is loaded in the CMU\_FDR. The frequency  $f_x^{(i)}$  can be derived from the value loaded in the CMU\_FDR as follows:

**Equation 1**      $f_x = (f_{FXOSC} \times MD) / n$

where n is the value in the CMU\_FDR and MD is the value in the CMU\_MDR.

The frequency meter by default evaluates  $f_{FIRC\_clk}$ , but software can swap to  $f_{SIRC\_clk}$  or  $f_{SXOSC\_clk}$  by programming the CKSEL bits in the CMU\_CSR.

### 6.8.5 Memory map and register description

The memory map of the CMU is shown in [Table 48](#).

**Table 48. CMU memory map**

Base address: 0xC3FE_0100			
Register name	Address offset	Reset value	Location
Control Status Register (CMU_CSR)	0x00	0x00000006	<a href="#">on page 160</a>
Frequency Display Register (CMU_FDR)	0x04	0x00000000	<a href="#">on page 161</a>
High Frequency Reference Register FMPLL (CMU_HFREFR)	0x08	0x00000FFF	<a href="#">on page 161</a>
Low Frequency Reference Register FMPLL (CMU_LFREFR)	0x0C	0x00000000	<a href="#">on page 162</a>
Interrupt Status Register (CMU_ISR)	0x10	0x00000000	<a href="#">on page 162</a>
Reserved	0x14	0x00000000	—
Measurement Duration Register (CMU_MDR)	0x18	0x00000000	<a href="#">on page 163</a>

i. x = FIRC,SIRC or SXOSC

6.8.5.1 Control Status Register (CMU\_CSR)

Offset: 0x00

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	0	0	0	0	0	0	0	0	SFM <sup>(1)</sup>	0	0	0	0	0	0	0	0
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CKSEL1		0	0	0	0	0	RCDIV		CME_A
W																
Reset	0	0	0	0	0	0	0		0	0	0	0	0	1	1	0

Figure 36. Control Status Register (CMU\_CSR)

1. You can read this field, and you can write a value of "1" to it. Writing a "0" has no effect. A reset will also clear this bit.

Table 49. CMU\_CSR field descriptions

Field	Description
SFM	Start frequency measure. The software can only set this bit to start a clock frequency measure. It is reset by hardware when the measure is ready in the CMU_FDR register. 0 Frequency measurement completed or not yet started. 1 Frequency measurement not completed.
CKSEL1	Clock oscillator selection bit. CKSEL1 selects the clock to be measured by the frequency meter. 00 FIRC_clk selected. 01 SIRC_clk selected. 10 SXOSC_clk selected. 11 FIRC_clk selected.
RCDIV	RC clock division factor . These bits specify the RC clock division factor. The output clock is FIRC_clk divided by the factor 2 <sup>RCDIV</sup> . This output clock is used to compare with FXOSC_clk for crystal clock monitor feature.The clock division coding is as follows. 00 Clock divided by 1 (No division) 01 Clock divided by 2 10 Clock divided by 4 11 Clock divided by 8
CME_A	FMPLL_0 clock monitor enable. 0 FMPLL_0 monitor disabled. 1 FMPLL_0 monitor enabled.



6.8.5.2 Frequency Display Register (CMU\_FDR)

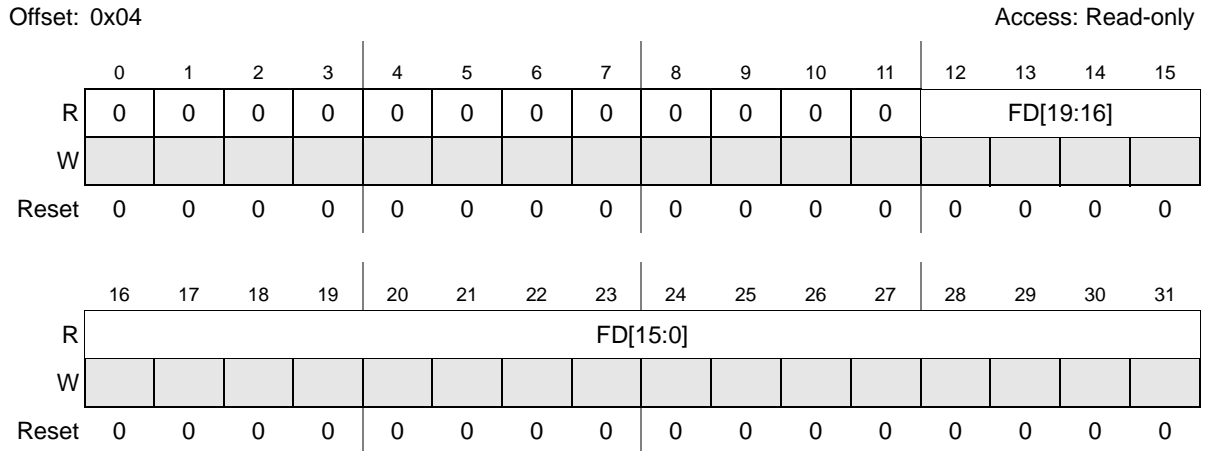


Figure 37. Frequency Display Register (CMU\_FDR)

Table 50. CMU\_FDR field descriptions

Field	Description
FD	Measured frequency bits. This register displays the measured frequency $f_x$ with respect to $f_{FXOSC}$ . The measured value is given by the following formula: $f_x = (f_{FXOSC} \times MD) / n$ , where n is the value in CMU_FDR register. Note: x = FIRC, SIRC or SXOSC.

6.8.5.3 High Frequency Reference Register FMPLL (CMU\_HFREFR)

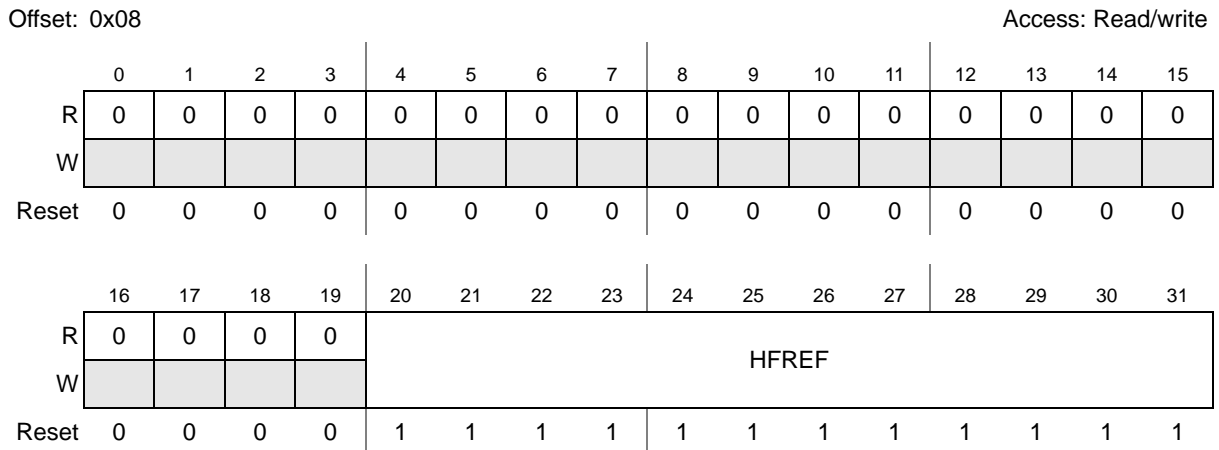


Figure 38. High Frequency Reference Register FMPLL (CMU\_HFREFR)

Table 51. CMU\_HFREFR field descriptions

Field	Description
HFREF	High Frequency reference value. This field determines the high reference value for the FMPLL clock. The reference value is given by: $(HFREF \div 16) \times (f_{FIRC} \div 4)$ .

6.8.5.4 Low Frequency Reference Register FMPLL (CMU\_LFREFR)

Offset: 0x0C Access: Read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	LFREF											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 39. Low Frequency Reference Register FMPLL (CMU\_LFREFR)

Table 52. CMU\_LFREFR field descriptions

Field	Description
LFREF	Low Frequency reference value. This field determines the low reference value for the FMPLL. The reference value is given by: $(LFREF \div 16) \times (f_{FIRC} \div 4)$ .

6.8.5.5 Interrupt Status Register (CMU\_ISR)

Offset: 0x10 Access: Read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	FHH	FLI	OLRI
W													w1c	w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 40. Interrupt status register (CMU\_ISR)

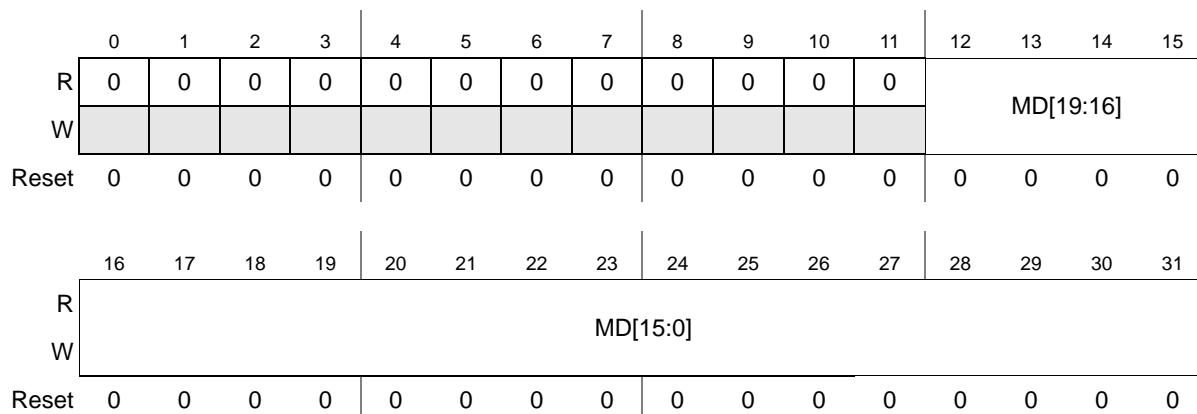
**Table 53. CMU\_ISR field descriptions**

Field	Description
FHHI	FMPLL clock frequency higher than high reference interrupt. This bit is set by hardware when $f_{FMPLL\_clk}$ becomes higher than HFREF value and FMPLL_clk is 'ON' as signalled by the MC_ME. It can be cleared by software by writing '1'. 0 No FHH event. 1 FHH event is pending.
FLLI	FMPLL clock frequency lower than low reference event. This bit is set by hardware when $f_{FMPLL\_clk}$ becomes lower than LFREF value and FMPLL_clk is 'ON' as signalled by the MC_ME. It can be cleared by software by writing '1'. 0 No FLL event. 1 FLL event is pending.
OLRI	Oscillator frequency lower than RC frequency event. This bit is set by hardware when $f_{FXOSC\_clk}$ is lower than $FIRC\_clk/2^{RCDIV}$ frequency and FXOSC_clk is 'ON' as signalled by the MC_ME. It can be cleared by software by writing '1'. 0 No OLR event. 1 OLR event is pending.

**6.8.5.6 Measurement Duration Register (CMU\_MDR)**

Offset: 0x18

Access: Read/write



**Figure 41. Measurement Duration Register (CMU\_MDR)**

**Table 54. CMU\_MDR field descriptions**

Field	Description
MD	Measurement duration bits. This field displays the measurement duration in numbers of clock cycles of the selected clock source. This value is loaded in the frequency meter downcounter. When CMU_CSR[SFM] = 1, the downcounter starts counting.

## 7 Clock Generation Module (MC\_CGM)

### 7.1 Overview

The clock generation module (MC\_CGM) generates reference clocks for all SoC blocks. The MC\_CGM selects one of the system clock sources to supply the system clock. The MC\_ME controls the system clock selection (see the MC\_ME chapter for more details). A set of MC\_CGM registers controls the clock dividers which are utilized for divided system and peripheral clock generation. The memory spaces of system and peripheral clock sources which have addressable memory spaces, are accessed through the MC\_CGM memory space. The MC\_CGM also selects and generates an output clock.

*Figure 42* depicts the MC\_CGM block diagram.

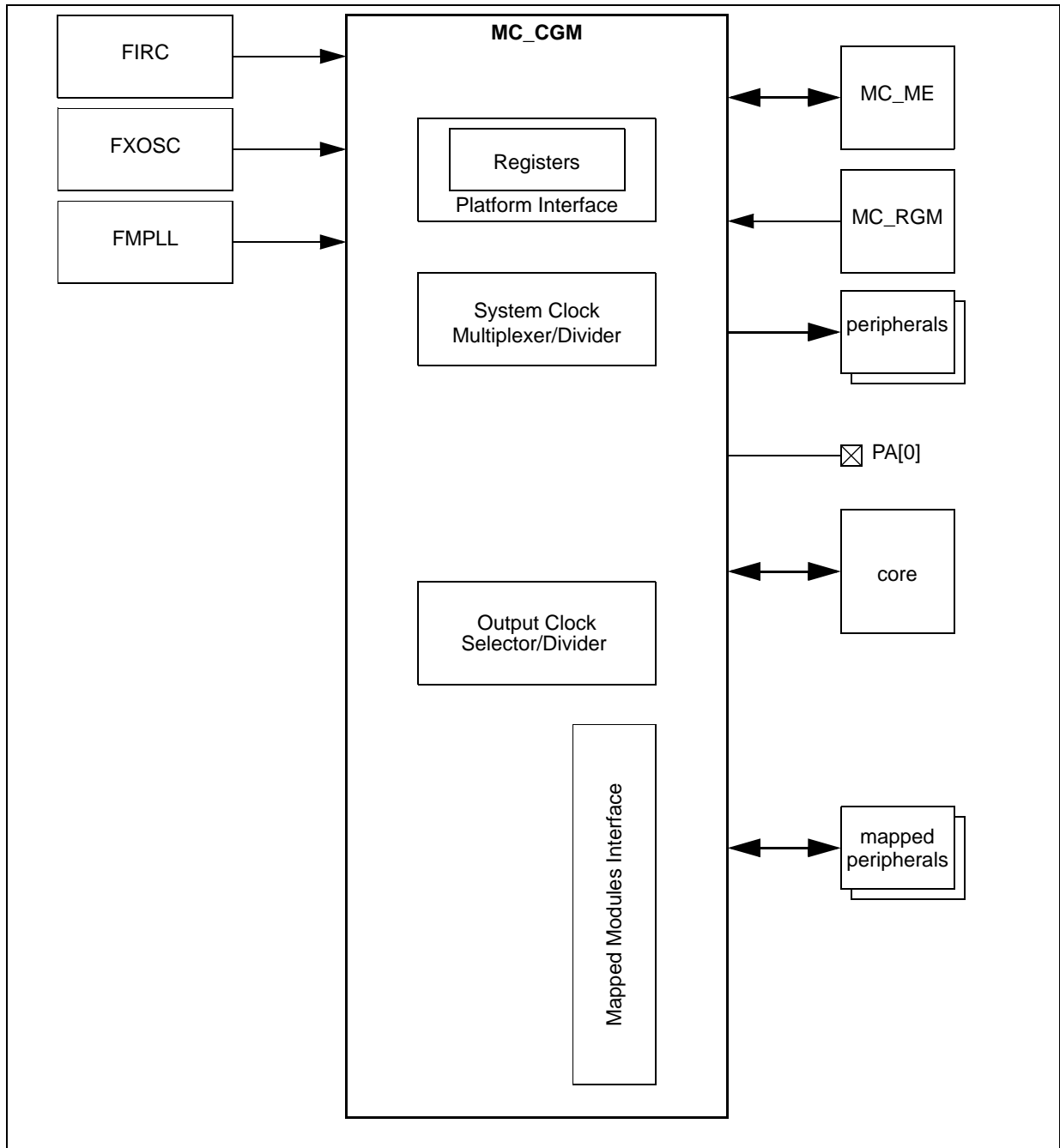


Figure 42. MC\_CGM block diagram

## 7.2 Features

The MC\_CGM includes the following features:

- generates system and peripheral clocks
- selects and enables/disables the system clock supply from system clock sources according to MC\_ME control
- contains a set of registers to control clock dividers for divided clock generation
- supports multiple clock sources and maps their address spaces to its memory map
- generates an output clock
- guarantees glitch-less clock transitions when changing the system clock selection
- supports 8-, 16- and 32-bit wide read/write accesses

## 7.3 Modes of operation

This section describes the basic functional modes of the MC\_CGM.

### 7.3.1 Normal and reset modes of operation

During normal and reset modes of operation, the clock selection for the system clock is controlled by the MC\_ME.

## 7.4 External signal description

The MC\_CGM delivers an output clock to the PA[0] pin for off-chip use and/or observation.

## 7.5 Memory map and register definition

Table 55. MC\_CGM register description

Address	Name	Description	Size	Access			Location
				Normal	Supervisor	Test	
0xC3FE_0370	CGM_OC_EN	Output Clock Enable	word	read	read/write	read/write	<a href="#">on page 171</a>
0xC3FE_0374	CGM_OCDS_SC	Output Clock Division Select	byte	read	read/write	read/write	<a href="#">on page 171</a>
0xC3FE_0378	CGM_SC_SS	System Clock Select Status	byte	read	read	read	<a href="#">on page 172</a>
0xC3FE_037C	CGM_SC_DC0	System Clock Divider Configuration 0	byte	read	read/write	read/write	<a href="#">on page 173</a>
0xC3FE_037D	CGM_SC_DC1	System Clock Divider Configuration 1	byte	read	read/write	read/write	<a href="#">on page 173</a>
0xC3FE_037E	CGM_SC_DC2	System Clock Divider Configuration 2	byte	read	read/write	read/write	<a href="#">on page 173</a>

Note: Any access to unused registers as well as write accesses to read-only registers will:  
 not change register content  
 cause a transfer error

Table 56. MC\_CGM memory map

Address	Name	0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0000 ... 0xC3FE_001C		FXOSC registers															
0xC3FE_0020 ... 0xC3FE_003C		reserved															
0xC3FE_0040 ... 0xC3FE_005C		SXOSC registers															
0xC3FE_0060 ... 0xC3FE_007C		FIRC registers															
0xC3FE_0080 ... 0xC3FE_009C		SIRC registers															
0xC3FE_00A0 ... 0xC3FE_00BC		FMPLL registers															
0xC3FE_00C0 ... 0xC3FE_00DC		reserved															
0xC3FE_00E0 ... 0xC3FE_00FC		reserved															

Table 56. MC\_CGM memory map (continued)

Address	Name	0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0100 ... 0xC3FE_011C	CMU registers																
0xC3FE_0120 ... 0xC3FE_013C	reserved																
0xC3FE_0140 ... 0xC3FE_015C	reserved																
0xC3FE_0160 ... 0xC3FE_017C	reserved																
0xC3FE_0180 ... 0xC3FE_019C	reserved																
0xC3FE_01A0 ... 0xC3FE_01BC	reserved																
0xC3FE_01C0 ... 0xC3FE_01DC	reserved																
0xC3FE_01E0 ... 0xC3FE_01FC	reserved																
0xC3FE_0200 ... 0xC3FE_021C	reserved																



Table 56. MC\_CGM memory map (continued)

Address	Name	0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0220 ... 0xC3FE_023C		reserved															
0xC3FE_0240 ... 0xC3FE_025C		reserved															
0xC3FE_0260 ... 0xC3FD_C27C		reserved															
0xC3FE_0280 ... 0xC3FE_029C		reserved															
0xC3FE_02A0 ... 0xC3FE_02BC		reserved															
0xC3FE_02C0 ... 0xC3FE_02DC		reserved															
0xC3FE_02E0 ... 0xC3FE_02FC		reserved															
0xC3FE_0300 ... 0xC3FE_031C		reserved															
0xC3FE_0320 ... 0xC3FE_033C		reserved															

Table 56. MC\_CGM memory map (continued)

Address	Name	Bit positions																	
		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15		
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
0xC3FE_0340 ... 0xC3FE_035C		reserved																	
0xC3FE_0360 ... 0xC3FE_036C		reserved																	
0xC3FE_0370	CGM_OC_EN	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
		W																	
0xC3FE_0374	CGM_OCDS_SC	R	0	0	SELDIV			SELCTL			0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																	
0xC3FE_0378	CGM_SC_SS	R	0	0	0	0	SELSTAT			0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																	
0xC3FE_037C	CGM_SC_DC 0...2	R	DE0	0	0	0	DIV0			DE1	0	0	0	DIV1					
		W																	
		R	DE2	0	0	0	DIV2			0	0	0	0	0	0	0	0	0	
		W																	
0xC3FE_0400 ... 0xC3FE_3FFC		reserved																	

### 7.5.1 Register descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the CGM\_OC\_EN register may be accessed as a word at address 0xC3FE\_0370, as a half-word at address 0xC3FE\_0372, or as a byte at address 0xC3FE\_0373.



**7.5.1.1 Output Clock Enable Register (CGM\_OC\_EN)**

Address 0xC3FE\_0370 Access: User read, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 43. Output Clock Enable Register (CGM\_OC\_EN)**

This register is used to enable and disable the output clock.

**Table 57. Output Clock Enable Register (CGM\_OC\_EN) field descriptions**

Field	Description
EN	Output Clock Enable control 0 Output Clock is disabled 1 Output Clock is enabled

**7.5.2 Output Clock Division Select Register (CGM\_OCDS\_SC)**

Address 0xC3FE\_0374 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SELDIV		SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 44. Output Clock Division Select Register (CGM\_OCDS\_SC)**

This register is used to select the current output clock source and by which factor it is divided before being delivered at the output clock.

**Table 58. Output Clock Division Select Register (CGM\_OCDS\_SC) field descriptions**

Field	Description
SELDIV	Output Clock Division Select 00 output selected Output Clock without division 01 output selected Output Clock divided by 2 10 output selected Output Clock divided by 4 11 output selected Output Clock divided by 8
SELCTL	<b>Output Clock Source Selection Control</b> — This value selects the current source for the output clock. 0000 4-16 MHz ext. xtal osc. 0001 16 MHz int. RC osc. 0010 freq. mod. PLL 0011 system clock 0100 RTC clock 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

### 7.5.3 System Clock Select Status Register (CGM\_SC\_SS)

This register provides the current clock source selection for the following clocks:

- undivided: system clock
- divided by system clock divider 0: peripheral set 1 clock
- divided by system clock divider 1: peripheral set 2 clock
- divided by system clock divider 2: peripheral set 3 clock

See [Figure 45](#) for details.

Address 0xC3FE\_0378 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELSTAT				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 45. System Clock Select Status Register (CGM\_SC\_SS)**

**Table 59. System Clock Select Status Register (CGM\_SC\_SS) field descriptions**

Field	Description
SELSTAT	<b>System Clock Source Selection Status</b> — This value indicates the current source for the system clock.
	0000 16 MHz int. RC osc.
	0001 div. 16 MHz int. RC osc.
	0010 4-16 MHz ext. xtal osc.
	0011 div. ext. xtal osc.
	0100 freq. mod. PLL
	0101 reserved
	0110 reserved
	0111 reserved
	1000 reserved
	1001 reserved
	1010 reserved
	1011 reserved
	1100 reserved
	1101 reserved
1110 reserved	
1111 system clock is disabled	

**7.5.3.1 System Clock Divider Configuration Registers (CGM\_SC\_DC0...2)**

Address 0xC3FE\_037C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DE0	0	0	0	DIV0				DE1	0	0	0	DIV1			
W																
Reset	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DE2	0	0	0	DIV2				0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 46. System Clock Divider Configuration Registers (CGM\_SC\_DC0...2)**

These registers control the system clock dividers.

**Table 60. System Clock Divider Configuration Registers (CGM\_SC\_DC0...2) Field Descriptions**

Field	Description
DE0	Divider 0 Enable 0 Disable system clock divider 0 1 Enable system clock divider 0
DIV0	Divider 0 Division Value — The resultant peripheral set 1 clock will have a period DIV0 + 1 times that of the system clock. If the DE0 is set to '0' (Divider 0 is disabled), any write access to the DIV0 field is ignored and the peripheral set 1 clock remains disabled.

**Table 60. System Clock Divider Configuration Registers (CGM\_SC\_DC0...2)  
Field Descriptions (continued)**

Field	Description
DE1	Divider 1 Enable 0 Disable system clock divider 1 1 Enable system clock divider 1
DIV1	Divider 1 Division Value — The resultant peripheral set 2 clock will have a period DIV1 + 1 times that of the system clock. If the DE1 is set to '0' (Divider 1 is disabled), any write access to the DIV1 field is ignored and the peripheral set 2 clock remains disabled.
DE2	Divider 2 Enable 0 Disable system clock divider 2 1 Enable system clock divider 2
DIV2	Divider 2 Division Value — The resultant peripheral set 3 clock will have a period DIV2 + 1 times that of the system clock. If the DE2 is set to '0' (Divider 2 is disabled), any write access to the DIV2 field is ignored and the peripheral set 3 clock remains disabled.

## 7.6 Functional Description

### 7.6.1 System Clock Generation

*Figure 47* shows the block diagram of the system clock generation logic. The MC\_ME provides the system clock select and switch mask (see MC\_ME chapter for more details), and the MC\_RGM provides the safe clock request (see MC\_RGM chapter for more details). The safe clock request forces the selector to select the 16 MHz int. RC osc. as the system clock and to ignore the system clock select.

#### 7.6.1.1 System Clock Source Selection

During normal operation, the system clock selection is controlled

- on a **SAFE** mode or reset event, by the MC\_RGM
- otherwise, by the MC\_ME

#### 7.6.1.2 System Clock Disable

During normal operation, the system clock can be disabled by the MC\_ME.

#### 7.6.1.3 System Clock Dividers

The MC\_CGM generates three derived clocks from the system clock.

#### 7.6.1.4 Dividers Functional Description

Dividers are utilized for the generation of divided system and peripheral clocks. The MC\_CGM has the following control registers for built-in dividers:

- [Section 7.5.3.1: System Clock Divider Configuration Registers \(CGM\\_SC\\_DC0...2\)](#)

The reset value of all counters is '1'. If a divider has its DE bit in the respective configuration register set to '0' (the divider is disabled), any value in its DIVn field is ignored

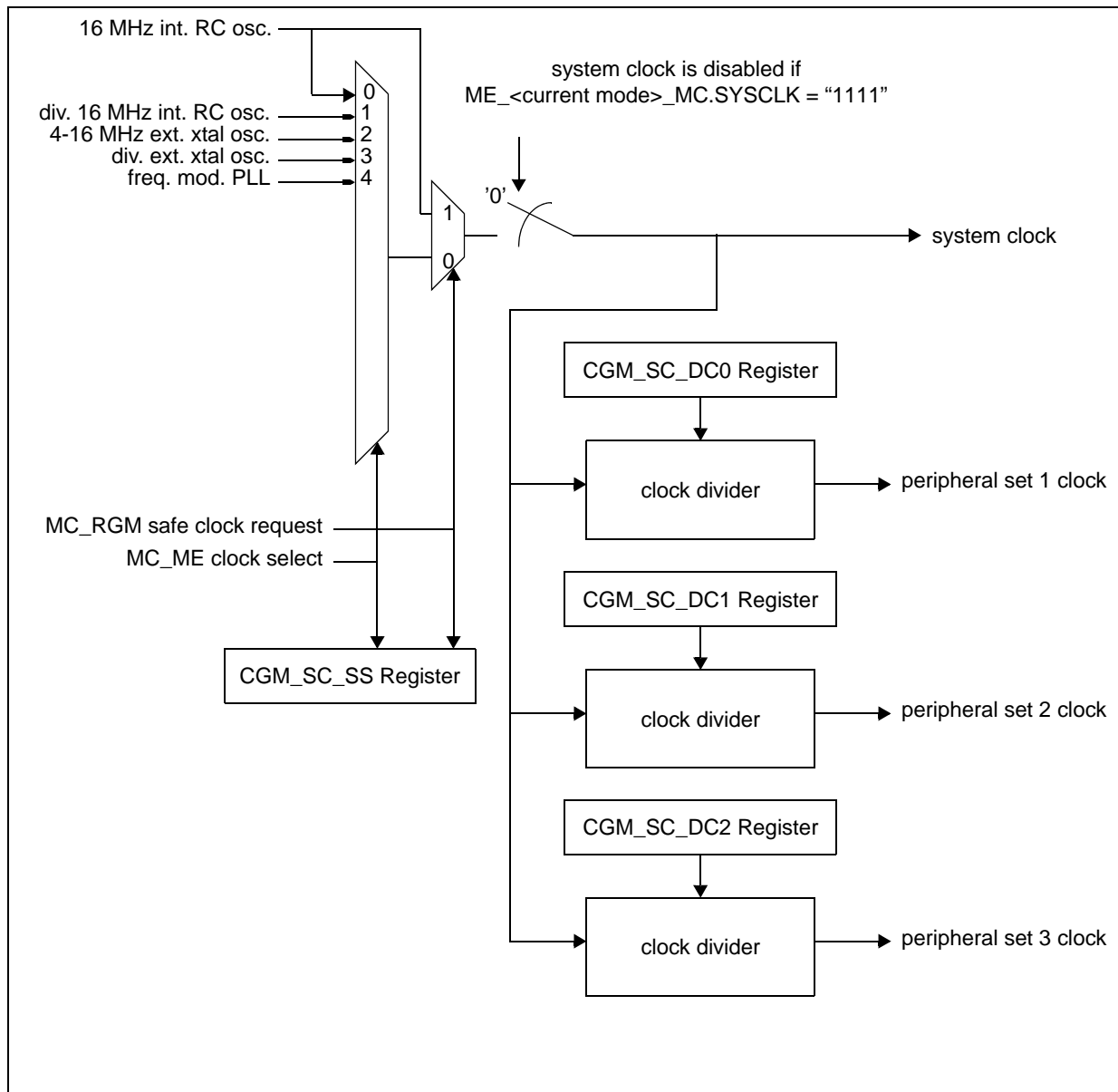


Figure 47. MC\_CGM System Clock Generation Overview

### 7.6.2 Output Clock Multiplexing

The MC\_CGM contains a multiplexing function for a number of clock sources which can then be utilized as output clock sources. The selection is done via the CGM\_OCDS\_SC register.

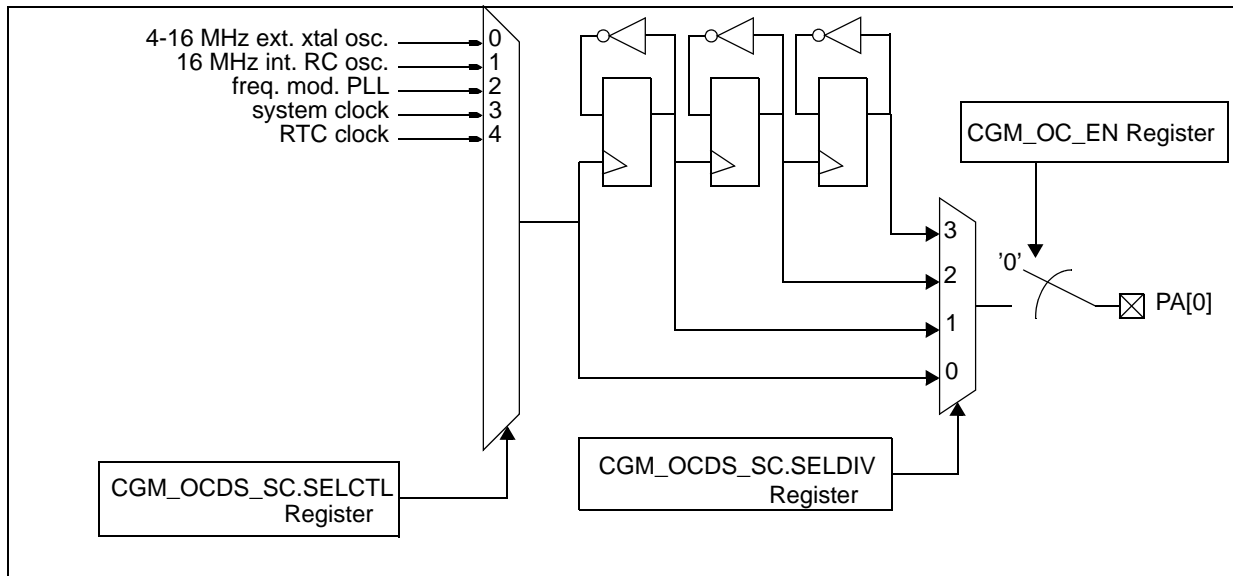


Figure 48. MC\_CGM Output Clock Multiplexer and PA[0] Generation

### 7.6.3 Output Clock Division Selection

The MC\_CGM provides the following output signals for the output clock generation:

- PA[0] (see [Figure 48](#)). This signal is generated by utilizing one of the 3-stage ripple counter outputs or the selected signal without division. The non-divided signal is not guaranteed to be 50% duty cycle by the MC\_CGM.
- the MC\_CGM also has an output clock enable register (see [Section 7.5.1.1: Output Clock Enable Register \(CGM\\_OC\\_EN\)](#)) which contains the output clock enable/disable control bit.



## 8 Mode Entry Module (MC\_ME)

### 8.1 Overview

The MC\_ME controls the SoC mode and mode transition sequences in all functional states. It also contains configuration, control and status registers accessible for the application.

*Figure 49* depicts the MC\_ME block diagram.

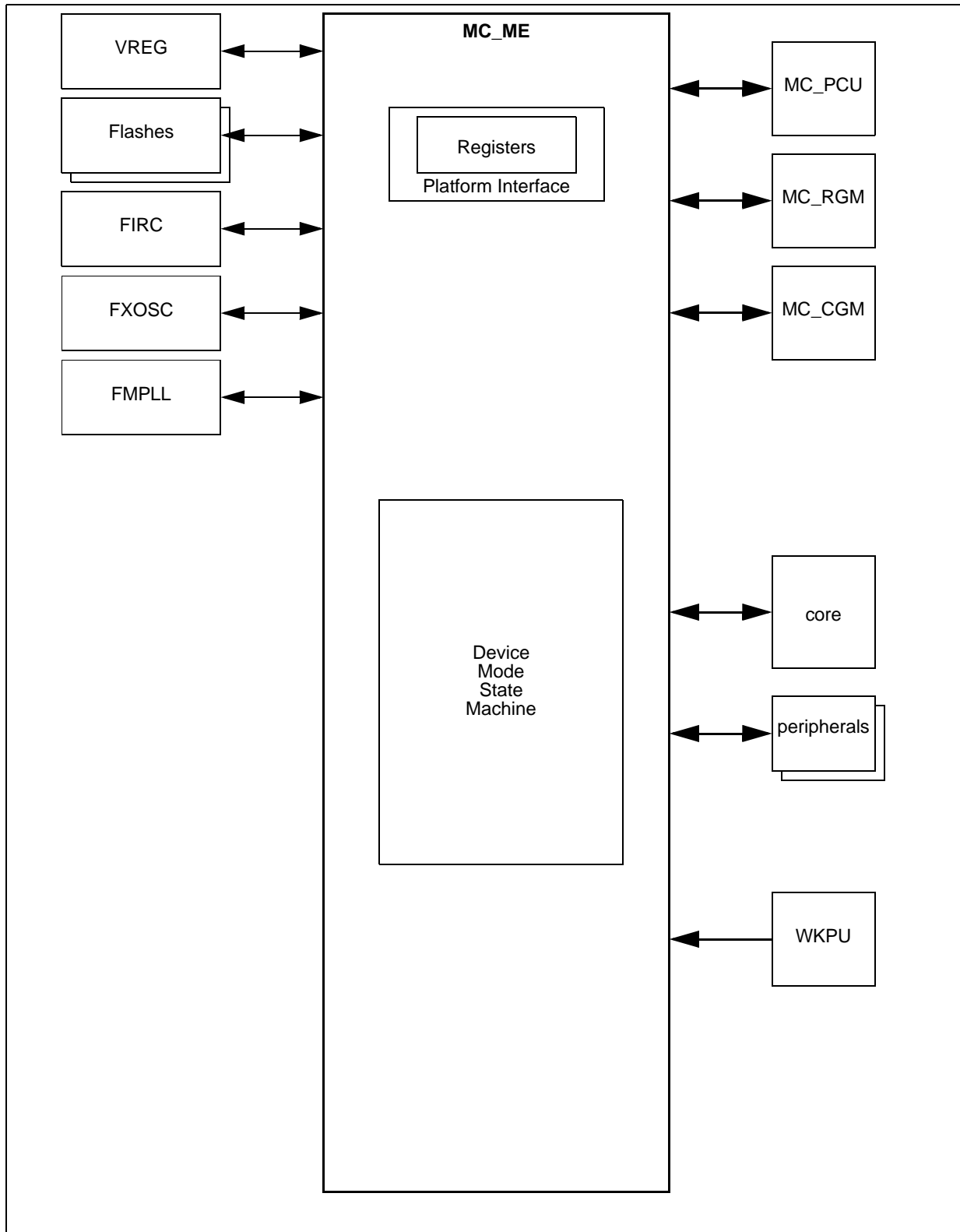


Figure 49. MC\_ME block diagram

### 8.1.1 Features

The MC\_ME includes the following features:

- control of the available modes by the ME\_ME register
- definition of various device mode configurations by the ME\_<mode>\_MC registers
- control of the actual device mode by the ME\_MCTL register
- capture of the current mode and various resource status within the contents of the ME\_GS register
- optional generation of various mode transition interrupts
- status bits for each cause of invalid mode transitions
- peripheral clock gating control based on the ME\_RUN\_PC0...7, ME\_LP\_PC0...7, and ME\_PCTL0...143 registers
- capture of current peripheral clock gated/enabled status

### 8.1.2 Modes of Operation

The MC\_ME is based on several device modes corresponding to different usage models of the device. Each mode is configurable and can define a policy for energy and processing power management to fit particular system requirements. An application can easily switch from one mode to another depending on the current needs of the system. The operating modes controlled by the MC\_ME are divided into system and user modes. The system modes are modes such as RESET, DRUN, SAFE, and TEST. These modes aim to ease the configuration and monitoring of the system. The user modes are modes such as RUN0...3, HALT, STOP, and STANDBY which can be configured to meet the application requirements in terms of energy management and available processing power. The modes DRUN, SAFE, TEST, and RUN0...3 are the device software running modes.

[Table 61](#) describes the MC\_ME modes.

**Table 61. MC\_ME Mode Descriptions**

Name	Description	Entry	Exit
RESET	This is a chip-wide virtual mode during which the application is not active. The system remains in this mode until all resources are available for the embedded software to take control of the device. It manages hardware initialization of chip configuration, voltage regulators, oscillators, PLLs, and flash modules.	system reset assertion from MC_RGM	system reset deassertion from MC_RGM
DRUN	This is the entry mode for the embedded software. It provides full accessibility to the system and enables the configuration of the system at startup. It provides the unique gate to enter USER modes. BAM when present is executed in DRUN mode.	system reset deassertion from MC_RGM, software request from SAFE, TEST and RUN0...3, wakeup request from STANDBY	system reset assertion, RUN0...3, TEST, STANDBY via software, SAFE via software or hardware failure.
SAFE	This is a chip-wide service mode which may be entered on the detection of a recoverable error. It forces the system into a pre-defined safe configuration from which the system may try to recover.	hardware failure, software request from DRUN, TEST, and RUN0...3	system reset assertion, DRUN via software

**Table 61. MC\_ME Mode Descriptions (continued)**

Name	Description	Entry	Exit
TEST	This is a chip-wide service mode which is intended to provide a control environment for device self-test. It may enable the application to run its own self-test like flash checksum, memory BIST etc.	software request from DRUN	system reset assertion, DRUN via software
RUN0...3	These are software running modes where most processing activity is done. These various run modes allow to enable different clock & power configurations of the system with respect to each other.	software request from DRUN, interrupt event from HALT, interrupt or wakeup event from STOP	system reset assertion, SAFE via software or hardware failure, other RUN0...3 modes, HALT, STOP, STANDBY via software
HALT	This is a reduced-activity low-power mode during which the clock to the core is disabled. It can be configured to switch off analog peripherals like PLL, flash, main regulator etc. for efficient power management at the cost of higher wakeup latency.	software request from RUN0...3	system reset assertion, SAFE on hardware failure, RUN0...3 on interrupt event
STOP	This is an advanced low-power mode during which the clock to the core is disabled. It may be configured to switch off most of the peripherals including oscillator for efficient power management at the cost of higher wakeup latency.	software request from RUN0...3	system reset assertion, SAFE on hardware failure, RUN0...3 on interrupt event or wakeup event
STANDBY	This is a reduced-leakage low-power mode during which power supply is cut off from most of the device. Wakeup from this mode takes a relatively long time, and content is lost or must be restored from backup.	software request from RUN0...3, DRUN modes	system reset assertion, DRUN on wakeup event

## 8.2 External Signal Description

The MC\_ME has no connections to any external pins.

## 8.3 Memory Map and Register Definition

The MC\_ME contains registers for:

- mode selection and status reporting
- mode configuration
- mode transition interrupts status and mask control
- scalable number of peripheral sub-mode selection and status reporting

Table 62. MC\_ME Register Description

Address	Name	Description	Size	Access			Location
				Normal	Supervisor	Test	
0xC3FD_C000	ME_GS	Global Status	word	read	read	read	<a href="#">on page 189</a>
0xC3FD_C004	ME_MCTL	Mode Control	word	read	read/write	read/write	<a href="#">on page 191</a>
0xC3FD_C008	ME_ME	Mode Enable	word	read	read/write	read/write	<a href="#">on page 192</a>
0xC3FD_C00C	ME_IS	Interrupt Status	word	read	read/write	read/write	<a href="#">on page 194</a>
0xC3FD_C010	ME_IM	Interrupt Mask	word	read	read/write	read/write	<a href="#">on page 195</a>
0xC3FD_C014	ME_IMTS	Invalid Mode Transition Status	word	read	read/write	read/write	<a href="#">on page 196</a>
0xC3FD_C018	ME_DMTS	Debug Mode Transition Status	word	read	read	read	<a href="#">on page 197</a>
0xC3FD_C020	ME_RESET_MC	RESET Mode Configuration	word	read	read	read	<a href="#">on page 199</a>
0xC3FD_C024	ME_TEST_MC	TEST Mode Configuration	word	read	read/write	read/write	<a href="#">on page 200</a>
0xC3FD_C028	ME_SAFE_MC	SAFE Mode Configuration	word	read	read/write	read/write	<a href="#">on page 200</a>
0xC3FD_C02C	ME_DRUN_MC	DRUN Mode Configuration	word	read	read/write	read/write	<a href="#">on page 201</a>
0xC3FD_C030	ME_RUN0_MC	RUN0 Mode Configuration	word	read	read/write	read/write	<a href="#">on page 201</a>
0xC3FD_C034	ME_RUN1_MC	RUN1 Mode Configuration	word	read	read/write	read/write	<a href="#">on page 201</a>
0xC3FD_C038	ME_RUN2_MC	RUN2 Mode Configuration	word	read	read/write	read/write	<a href="#">on page 201</a>
0xC3FD_C03C	ME_RUN3_MC	RUN3 Mode Configuration	word	read	read/write	read/write	<a href="#">on page 201</a>
0xC3FD_C040	ME_HALT_MC	HALT Mode Configuration	word	read	read/write	read/write	<a href="#">on page 202</a>
0xC3FD_C048	ME_STOP_MC	STOP Mode Configuration	word	read	read/write	read/write	<a href="#">on page 202</a>
0xC3FD_C054	ME_STANDBY_MC	STANDBY Mode Configuration	word	read	read/write	read/write	<a href="#">on page 203</a>
0xC3FD_C060	ME_PS0	Peripheral Status 0	word	read	read	read	<a href="#">on page 205</a>
0xC3FD_C064	ME_PS1	Peripheral Status 1	word	read	read	read	<a href="#">on page 205</a>
0xC3FD_C068	ME_PS2	Peripheral Status 2	word	read	read	read	<a href="#">on page 206</a>
0xC3FD_C06C	ME_PS3	Peripheral Status 3	word	read	read	read	<a href="#">on page 206</a>
0xC3FD_C080	ME_RUN_PC0	Run Peripheral Configuration 0	word	read	read/write	read/write	<a href="#">on page 207</a>
0xC3FD_C084	ME_RUN_PC1	Run Peripheral Configuration 1	word	read	read/write	read/write	<a href="#">on page 207</a>

Table 62. MC\_ME Register Description (continued)

Address	Name	Description	Size	Access			Location
				Normal	Supervisor	Test	
...							
0xC3FD_C09C	ME_RUN_PC7	Run Peripheral Configuration 7	word	read	read/write	read/write	<a href="#">on page 207</a>
0xC3FD_C0A0	ME_LP_PC0	Low-Power Peripheral Configuration 0	word	read	read/write	read/write	<a href="#">on page 208</a>
0xC3FD_C0A4	ME_LP_PC1	Low-Power Peripheral Configuration 1	word	read	read/write	read/write	<a href="#">on page 208</a>
...							
0xC3FD_C0BC	ME_LP_PC7	Low-Power Peripheral Configuration 7	word	read	read/write	read/write	<a href="#">on page 208</a>
0xC3FD_C0C4	ME_PCTL4	DSPI0 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0C5	ME_PCTL5	DSPI1 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0C6	ME_PCTL6	DSPI2 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0C7	ME_PCTL7	DSPI3 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0C8	ME_PCTL8	DSPI4 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0C9	ME_PCTL9	DSPI5 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0CC	ME_PCTL12	LINFlex8 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0CD	ME_PCTL13	LINFlex9 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0D0	ME_PCTL16	FlexCAN0 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0D1	ME_PCTL17	FlexCAN1 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0D2	ME_PCTL18	FlexCAN2 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0D3	ME_PCTL19	FlexCAN3 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0D4	ME_PCTL20	FlexCAN4 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0D5	ME_PCTL21	FlexCAN5 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0D7	ME_PCTL23	DMA_CH_MUX Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0E0	ME_PCTL32	ADC0 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0E1	ME_PCTL33	ADC1 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0EC	ME_PCTL44	I2C0 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F0	ME_PCTL48	LINFlex0 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F1	ME_PCTL49	LINFlex1 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F2	ME_PCTL50	LINFlex2 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F3	ME_PCTL51	LINFlex3 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>

Table 62. MC\_ME Register Description (continued)

Address	Name	Description	Size	Access			Location
				Normal	Supervisor	Test	
0xC3FD_C0F4	ME_PCTL52	LINFlex4 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F5	ME_PCTL53	LINFlex5 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F6	ME_PCTL54	LINFlex6 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F7	ME_PCTL55	LINFlex7 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0F9	ME_PCTL57	CTU Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C0FC	ME_PCTL60	CANSampler Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C104	ME_PCTL68	SIUL Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C105	ME_PCTL69	WKPU Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C108	ME_PCTL72	eMIOS0 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C109	ME_PCTL73	eMIOS1 Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C11B	ME_PCTL91	RTC_API Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C11C	ME_PCTL92	PIT_RTI Control	byte	read	read/write	read/write	<a href="#">on page 209</a>
0xC3FD_C128	ME_PCTL104	CMU Control	byte	read	read/write	read/write	<a href="#">on page 209</a>

Note: Any access to unused registers as well as write accesses to read-only registers will:  
 not change register content  
 cause a transfer error

Table 63. MC\_ME Memory Map

Address	Name	Bit Fields																
		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
0xC3FD_C000	ME_GS	R	S_CURRENT_MODE				S_MTRANS	S_DC	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
		W																
		R	0	0	0	0	0	0	0	0	0	S_FMPLL	S_FXOSC	S_FIRC	S_SYSCCLK			
		W																
0xC3FD_C004	ME_MCTL	R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	
		W																
		R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
		W	KEY															



Table 63. MC\_ME Memory Map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0xC3FD_C008	ME_ME	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	STANDBY	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET	
		W																	
0xC3FD_C00C	ME_IS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	0	0	0	0	0	I_ICONF	I_IMODE	I_SAFE	I_MTC	
		W													w1c	w1c	w1c	w1c	
0xC3FD_C010	ME_IM	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	0	0	0	0	0	M_ICONF	M_IMODE	M_SAFE	M_MTC	
		W																	
0xC3FD_C014	ME_IMTS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
		W													w1c	w1c	w1c	w1c	w1c
0xC3FD_C018	ME_DMTS	R	0	0	0	0	0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR	
		W																	
		R	0	FMPLL_SC	FXOSC_SC	FIRC_SC	0	SYCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	CDP_PRPH_96_127	CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31	
		W																	
0xC3FD_C01C	reserved																		





Table 63. MC\_ME Memory Map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0xC3FD_C020	ME_RESET_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON		
		W																	
		R	0	0	0	0	0	0	0	0	0		FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																	
0xC3FD_C024	ME_TEST_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON		
		W																	
		R	0	0	0	0	0	0	0	0	0		FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																	
0xC3FD_C028	ME_SAFE_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON		
		W																	
		R	0	0	0	0	0	0	0	0	0		FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																	
0xC3FD_C02C	ME_DRUN_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON		
		W																	
		R	0	0	0	0	0	0	0	0	0		FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																	

Table 63. MC\_ME Memory Map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C030 ... 0xC3FD_C03C	ME_RUN0...3_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C040	ME_HALT_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C044	reserved																	
0xC3FD_C048	ME_STOP_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																
0xC3FD_C04C ... 0xC3FD_C050	reserved																	
0xC3FD_C054	ME_STANDBY_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
		W																

Table 63. MC\_ME Memory Map (continued)

Address	Name	0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0xC3FD_C058 ... 0xC3FD_C05C	reserved																	
0xC3FD_C060	ME_PS0	R	0	0	0	0	0	0	0	0	S_DMA_CH_MUX	0	S_FlexCAN5	S_FlexCAN4	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
		W																
		R	0	0	S_LINFlex9	S_LINFlex8	0	0	S_DSPi5	S_DSPi4	S_DSPi3	S_DSPi2	S_DSPi1	S_DSPi0	0	0	0	0
		W																
0xC3FD_C064	ME_PS1	R	0	0	0	S_CANSampler	0	0	S_CTU	0	S_LINFlex7	S_LINFlex6	S_LINFlex5	S_LINFlex4	S_LINFlex3	S_LINFlex2	S_LINFlex1	S_LINFlex0
		W																
		R	0	0	0	S_I2C0	0	0	0	0	0	0	0	0	0	0	S_ADC1	S_ADC0
		W																
0xC3FD_C068	ME_PS2	R	0	0	0	S_PIT_RTI	S_RTC_API	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	S_eMIOS1	S_eMIOS0	0	0	S_WKPU	S_SIUL	0	0	0	0
		W																

Table 63. MC\_ME Memory Map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0xC3FD_C06C	ME_PS3	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	S_CMU	0	0	0	0	0	0	0	0	0
		W																	
0xC3FD_C070	reserved																		
0xC3FD_C074 ... 0xC3FD_C07C	reserved																		
0xC3FD_C080 ... 0xC3FD_C09C	ME_RUN_PC0...7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET	
		W																	
0xC3FD_C0A0 ... 0xC3FD_C0BC	ME_LP_PC0...7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																	
		R	0	0	STANDBY	0	0	STOP	0	HALT	0	0	0	0	0	0	0	0	0
		W																	
0xC3FD_C0C0 ... 0xC3FD_C14C	ME_PCTL0...143	R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG			
		W																	
		R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG			
		W																	
0xC3FD_C150 ... 0xC3FD_FFFC	reserved																		

### 8.3.1 Register Description

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the ME\_RUN\_PC0 register may be accessed as a word at address 0xC3FD\_C080, as a half-word at address 0xC3FD\_C082, or as a byte at address 0xC3FD\_C083.

8.3.1.1 Global Status Register (ME\_GS)

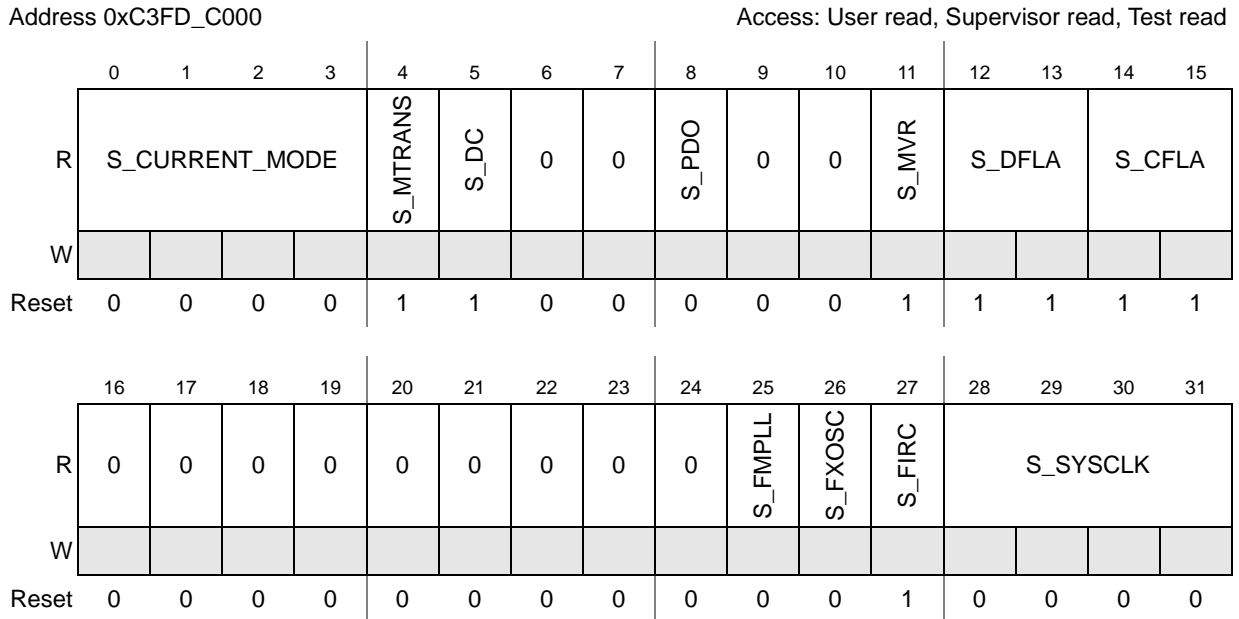


Figure 50. Global Status Register (ME\_GS)

This register contains global mode status.

Table 64. Global Status Register (ME\_GS) Field Descriptions

Field	Description
S_CURRENT_MODE	Current device mode status 0000 RESET 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 HALT 1001 reserved 1010 STOP 1011 reserved 1100 reserved 1101 STANDBY 1110 reserved 1111 reserved
S_MTRANS	Mode transition status 0 Mode transition process is not active 1 Mode transition is ongoing
S_DC	Device current consumption status 0 Device consumption is low enough to allow powering down of main voltage regulator 1 Device consumption requires main voltage regulator to remain powered regardless of mode configuration

**Table 64. Global Status Register (ME\_GS) Field Descriptions (continued)**

Field	Description
S_PDO	Output power-down status — This bit specifies output power-down status of I/Os. This bit is asserted whenever outputs of pads are forced to high impedance state or the pads power sequence driver is switched off. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled 1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In STOP mode, only pad power sequence driver is disabled but the state of the output is kept. In STANDBY mode, the power sequence driver and all pads except those mapped on wakeup lines are not powered and therefore high impedance. Wakeup lines configuration remains unchanged
S_MVR	Main voltage regulator status 0 Main voltage regulator is not ready 1 Main voltage regulator is ready for use
S_DFLA	Data flash availability status 00 Data flash is not available 01 Data flash is in power-down mode 10 Data flash is in low-power mode 11 Data flash is in normal mode and available for use
S_CFLA	Code flash availability status 00 Code flash is not available 01 Code flash is in power-down mode 10 Code flash is in low-power mode 11 Code flash is in normal mode and available for use
S_FMPLL	frequency modulated phase locked loop status 0 frequency modulated phase locked loop is not stable 1 frequency modulated phase locked loop is providing a stable clock
S_FXOSC	fast external crystal oscillator (4-16 MHz) status 0 fast external crystal oscillator (4-16 MHz) is not stable 1 fast external crystal oscillator (4-16 MHz) is providing a stable clock
S_FIRC	fast internal RC oscillator (16 MHz) status 0 fast internal RC oscillator (16 MHz) is not stable 1 fast internal RC oscillator (16 MHz) is providing a stable clock
S_SYSCLK	System clock switch status — These bits specify the system clock currently used by the system. 0000 16 MHz int. RC osc. 0001 div. 16 MHz int. RC osc. 0010 4-16 MHz ext. xtal osc. 0011 div. ext. xtal osc. 0100 freq. mod. PLL 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled

8.3.1.2 Mode Control Register (ME\_MCTL)

Address 0xC3FD\_C004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
W	KEY															
Reset	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1

Figure 51. Mode Control Register (ME\_MCTL)

This register is used to trigger software-controlled mode changes. Depending on the modes as enabled by ME\_ME register bits, configurations corresponding to unavailable modes are reserved and access to ME\_<mode>\_MC registers must respect this for successful mode requests.

*Note:* Byte and half-word write accesses are not allowed for this register as a predefined key is required to change its value.

Table 65. Mode Control Register (ME\_MCTL) Field Descriptions

Field	Description
TARGET_M ODE	<p><b>Target device mode</b> — These bits provide the target device mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: first time with key, and second time with inverted key. These bits are automatically updated by hardware while entering SAFE on hardware request. Also, while exiting from the HALT and STOP modes on hardware exit events, these are updated with the appropriate RUN0...3 mode value.</p> <p>0000 RESET                      0001 TEST                      0010 SAFE                      0011 DRUN                      0100 RUN0                      0101 RUN1                      0110 RUN2                      0111 RUN3                      1000 HALT                      1001 reserved                      1010 STOP                      1011 reserved                      1100 reserved                      1101 STANDBY                      1110 reserved                      1111 reserved</p>
KEY	<p><b>Control key</b> — These bits enable write access to this register. Any write access to the register with a value different from the keys is ignored. Read access will always return inverted key.</p> <p>KEY: 0101101011110000 (0x5AF0)                      INVERTED KEY: 1010010100001111 (0xA50F)</p>

8.3.1.3 Mode Enable Register (ME\_ME)

Address 0xC3FD\_C008 Access: User read, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STANDBY	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 52. Mode Enable Register (ME\_ME)

This register allows a way to disable the device modes which are not required for a given device. RESET, SAFE, DRUN, and RUN0 modes are always enabled.





Table 66. Mode Enable Register (ME\_ME) Field Descriptions

Field	Description
STANDBY	STANDBY mode enable 0 STANDBY mode is disabled 1 STANDBY mode is enabled
STOP	STOP mode enable 0 STOP mode is disabled 1 STOP mode is enabled
HALT	HALT mode enable 0 HALT mode is disabled 1 HALT mode is enabled
RUN3	RUN3 mode enable 0 RUN3 mode is disabled 1 RUN3 mode is enabled
RUN2	RUN2 mode enable 0 RUN2 mode is disabled 1 RUN2 mode is enabled
RUN1	RUN1 mode enable 0 RUN1 mode is disabled 1 RUN1 mode is enabled
RUN0	RUN0 mode enable 0 RUN0 mode is disabled 1 RUN0 mode is enabled
DRUN	DRUN mode enable 0 DRUN mode is disabled 1 DRUN mode is enabled
SAFE	SAFE mode enable 0 SAFE mode is disabled 1 SAFE mode is enabled
TEST	TEST mode enable 0 TEST mode is disabled 1 TEST mode is enabled
RESET	RESET mode enable 0 RESET mode is disabled 1 RESET mode is enabled

8.3.1.4 Interrupt Status Register (ME\_IS)

Address 0xC3FD\_C00C Access: User read, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 53. Interrupt Status Register (ME\_IS)

This register provides the current interrupt status.

Table 67. Interrupt Status Register (ME\_IS) Field Descriptions

Field	Description
I_ICONF	<b>Invalid mode configuration interrupt</b> — This bit is set whenever a write operation to ME_<mode>_MC registers with invalid mode configuration is attempted. It is cleared by writing a '1' to this bit. 0 No invalid mode configuration interrupt occurred 1 Invalid mode configuration interrupt is pending
I_IMODE	<b>Invalid mode interrupt</b> — This bit is set whenever an invalid mode transition is requested. It is cleared by writing a '1' to this bit. 0 No invalid mode interrupt occurred 1 Invalid mode interrupt is pending
I_SAFE	<b>SAFE mode interrupt</b> — This bit is set whenever the device enters SAFE mode on hardware requests generated in the system. It is cleared by writing a '1' to this bit. 0 No SAFE mode interrupt occurred 1 SAFE mode interrupt is pending
I_MTC	<b>Mode transition complete interrupt</b> — This bit is set whenever the mode transition process completes (S_MTRANS transits from 1 to 0). It is cleared by writing a '1' to this bit. This mode transition interrupt bit will not be set while entering low-power modes HALT, STOP, or STANDBY. 0 No mode transition complete interrupt occurred 1 Mode transition complete interrupt is pending

8.3.1.5 Interrupt Mask Register (ME\_IM)

Address 0xC3FD\_C010 Access: User read, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	M_ICONF	M_IMODE	M_SAFE	M_MTC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 54. Interrupt Mask Register (ME\_IM)

This register controls whether an event generates an interrupt or not.

Table 68. Interrupt Mask Register (ME\_IM) Field Descriptions

Field	Description
M_ICONF	<b>Invalid mode configuration interrupt mask</b> 0 Invalid mode interrupt is masked 1 Invalid mode interrupt is enabled
M_IMODE	<b>Invalid mode interrupt mask</b> 0 Invalid mode interrupt is masked 1 Invalid mode interrupt is enabled
M_SAFE	<b>SAFE mode interrupt mask</b> 0 SAFE mode interrupt is masked 1 SAFE mode interrupt is enabled
M_MTC	<b>Mode transition complete interrupt mask</b> 0 Mode transition complete interrupt is masked 1 Mode transition complete interrupt is enabled

8.3.1.6 Invalid Mode Transition Status Register (ME\_IMTS)

Address 0xC3FD\_C014 Access: User read, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 55. Invalid Mode Transition Status Register (ME\_IMTS)

This register provides the status bits for each cause of invalid mode interrupt.

Table 69. Invalid Mode Transition Status Register (ME\_IMTS) Field Descriptions

Field	Description
S_MTI	<b>Mode Transition Illegal status</b> — This bit is set whenever a new mode is requested while some other mode transition process is active (S_MTRANS is '1'). Please see <a href="#">Section 8.4.5: Mode Transition Interrupts</a> for the exceptions to this behavior. It is cleared by writing a '1' to this bit. 0 Mode transition requested is not illegal 1 Mode transition requested is illegal
S_MRI	<b>Mode Request Illegal status</b> — This bit is set whenever the target mode requested is not a valid mode with respect to current mode. It is cleared by writing a '1' to this bit. 0 Target mode requested is not illegal with respect to current mode 1 Target mode requested is illegal with respect to current mode
S_DMA	<b>Disabled Mode Access status</b> — This bit is set whenever the target mode requested is one of those disabled modes determined by ME_ME register. It is cleared by writing a '1' to this bit. 0 Target mode requested is not a disabled mode 1 Target mode requested is a disabled mode
S_NMA	<b>Non-existing Mode Access status</b> — This bit is set whenever the target mode requested is one of those non existing modes determined by ME_ME register. It is cleared by writing a '1' to this bit. 0 Target mode requested is an existing mode 1 Target mode requested is a non-existing mode
S_SEA	<b>SAFE Event Active status</b> — This bit is set whenever the device is in SAFE mode, SAFE event bit is pending and a new mode requested other than RESET/SAFE modes. It is cleared by writing a '1' to this bit. 0 No new mode requested other than RESET/SAFE while SAFE event is pending 1 New mode requested other than RESET/SAFE while SAFE event is pending

8.3.1.7 Debug Mode Transition Status Register (ME\_DMTS)

Address 0xC3FD\_C018 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	FMPLL_SC	FXOSC_SC	FIRC_SC	0	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	CDP_PRPH_96_127	CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 56. Debug Mode Transition Status Register (ME\_DMTS)

This register provides the status of different factors which influence mode transitions. It is used to give an indication of why a mode transition indicated by ME\_GS.S\_MTRANS may be taking longer than expected.

*Note:* The ME\_DMTS register does not indicate whether a mode transition is ongoing. Therefore, some ME\_DMTS bits may still be asserted after the mode transition has completed.

Table 70. Debug Mode Transition Status Register (ME\_DMTS) Field Descriptions

Field	Description
MPH_BUSY	MC_ME/MC_PCU Handshake Busy indicator — This bit is set if the MC_ME has requested a mode change from the MC_PCU and the MC_PCU has not yet responded. It is cleared when the MC_PCU has responded. 0 Handshake is not busy 1 Handshake is busy
PMC_PROG	MC_PCU Mode Change in Progress indicator — This bit is set if the MC_PCU is in the process of powering up or down power domains. It is cleared when all power-up/down processes have completed. 0 Power-up/down transition is not in progress 1 Power-up/down transition is in progress
CORE_DBG	Processor is in Debug mode indicator — This bit is set while the processor is in debug mode. 0 The processor is not in debug mode 1 The processor is in debug mode

**Table 70. Debug Mode Transition Status Register (ME\_DMTS) Field Descriptions (continued)**

Field	Description
SMR	SAFE mode request from MC_RGM is active indicator — This bit is set if a hardware SAFE mode request has been triggered. It is cleared when the hardware SAFE mode request has been cleared. 0 A SAFE mode request is not active 1 A SAFE mode request is active
FMPLL_SC	FMPLL State Change during mode transition indicator — This bit is set when the frequency modulated phase locked loop is requested to change its power up/down state. It is cleared when the frequency modulated phase locked loop has completed its state change. 0 No state change is taking place 1 A state change is taking place
FXOSC_SC	FXOSC State Change during mode transition indicator — This bit is set when the fast external crystal oscillator (4-16 MHz) is requested to change its power up/down state. It is cleared when the fast external crystal oscillator (4-16 MHz) has completed its state change. 0 No state change is taking place 1 A state change is taking place
FIRC_SC	FIRC State Change during mode transition indicator — This bit is set when the fast internal RC oscillator (16 MHz) is requested to change its power up/down state. It is cleared when the fast internal RC oscillator (16 MHz) has completed its state change. 0 No state change is taking place 1 A state change is taking place
SYSCLK_SW	System Clock Switching pending status — 0 No system clock source switching is pending 1 A system clock source switching is pending
DFLASH_SC	DFLASH State Change during mode transition indicator — This bit is set when the DFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place 1 A state change is taking place
CFLASH_SC	CFLASH State Change during mode transition indicator — This bit is set when the CFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place 1 A state change is taking place
CDP_PRPH_0_143	Clock Disable Process Pending status for Peripherals 0...143 — This bit is set when any peripheral has been requested to have its clock disabled. It is cleared when all the peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1 Clock disabling is pending for at least one peripheral
CDP_PRPH_96_127	Clock Disable Process Pending status for Peripherals 96...127 — This bit is set when any peripheral appearing in ME_PS3 has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1 Clock disabling is pending for at least one peripheral

**Table 70. Debug Mode Transition Status Register (ME\_DMTS) Field Descriptions (continued)**

Field	Description
CDP_PRPH_64_95	Clock Disable Process Pending status for Peripherals 64...95 — This bit is set when any peripheral appearing in ME_PS2 has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1 Clock disabling is pending for at least one peripheral
CDP_PRPH_32_63	Clock Disable Process Pending status for Peripherals 32...63 — This bit is set when any peripheral appearing in ME_PS1 has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1 Clock disabling is pending for at least one peripheral
CDP_PRPH_0_31	Clock Disable Process Pending status for Peripherals 0...31 — This bit is set when any peripheral appearing in ME_PS0 has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending 1 Clock disabling is pending for at least one peripheral

**8.3.1.8 RESET Mode Configuration Register (ME\_RESET\_MC)**

Address 0xC3FD\_C020

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MYRON	DFLAON	CFLAON		
W	[Greyed out]															
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
W	[Greyed out]															
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

**Figure 57. Invalid Mode Transition Status Register (ME\_IMTS)**

This register configures system behavior during RESET mode. Please see [Table 71](#) for details.

8.3.1.9 TEST Mode Configuration Register (ME\_TEST\_MC)

Address 0xC3FD\_C024 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W	[Greyed out]									[Greyed out]			[Greyed out]		[Greyed out]	
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLN	FXOSCON	FIRCON	SYSCLK			
W	[Greyed out]									[Greyed out]	[Greyed out]	[Greyed out]	[Greyed out]			
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 58. TEST Mode Configuration Register (ME\_TEST\_MC)

This register configures system behavior during TEST mode. Please see [Table 71](#) for details.

Note: Byte and half-word write accesses are not allowed to this register.

8.3.1.10 SAFE Mode Configuration Register (ME\_SAFE\_MC)

Address 0xC3FD\_C028 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W	[Greyed out]									[Greyed out]			[Greyed out]		[Greyed out]	
Reset	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLN	FXOSCON	FIRCON	SYSCLK			
W	[Greyed out]									[Greyed out]	[Greyed out]	[Greyed out]	[Greyed out]			
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 59. SAFE Mode Configuration Register (ME\_SAFE\_MC)

This register configures system behavior during SAFE mode. Please see [Table 71](#) for details.



Note: Byte and half-word write accesses are not allowed to this register.

**8.3.1.11 DRUN Mode Configuration Register (ME\_DRUN\_MC)**

Address 0xC3FD\_C02C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W	[Greyed out]															
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
W	[Greyed out]									FMPLLON	FXOSCON	[Greyed out]				
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 60. DRUN Mode Configuration Register (ME\_DRUN\_MC)

This register configures system behavior during DRUN mode. Please see [Table 71](#) for details.

Note: Byte and half-word write accesses are not allowed to this register.

Note: The values of FXOSCON, CFLAON and DFLAON are retained through STANDBY mode.

**8.3.1.12 RUN0...3 Mode Configuration Registers (ME\_RUN0...3\_MC)**

Address 0xC3FD\_C030 - 0xC3FD\_C03C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W	[Greyed out]															
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
W	[Greyed out]									FMPLLON	FXOSCON	[Greyed out]				
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 61. RUN0...3 Mode Configuration Registers (ME\_RUN0...3\_MC)



This register configures system behavior during RUN0...3 modes. Please see [Table 71](#) for details.

Note: Byte and half-word write accesses are not allowed to this register.

### 8.3.1.13 HALT Mode Configuration Register (ME\_HALT\_MC)

Address 0xC3FD\_C040 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 62. HALT Mode Configuration Register (ME\_HALT\_MC)

This register configures system behavior during HALT mode. Please see [Table 71](#) for details.

Note: Byte and half-word write accesses are not allowed to this register.

### 8.3.1.14 STOP Mode Configuration Register (ME\_STOP\_MC)

Address 0xC3FD\_C048 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 63. STOP Mode Configuration Register (ME\_STOP\_MC)

This register configures system behavior during STOP mode. Please see [Table 71](#) for details.

Note: Byte and half-word write accesses are not allowed to this register.

### 8.3.1.15 STANDBY Mode Configuration Register (ME\_STANDBY\_MC)

Address 0xC3FD\_C054

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON	CFLAON		
W	[Greyed out]															
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLN	FXOSCON	FIRCON	SYSCLK			
W	[Greyed out]															
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Figure 64. STANDBY Mode Configuration Register (ME\_STANDBY\_MC)

This register configures system behavior during STANDBY mode. Please see [Table 71](#) for details.

Note: Byte and half-word write accesses are not allowed to this register.

Table 71. Mode Configuration Registers (ME\_<mode>\_MC) Field Descriptions

Field	Description
PDO	I/O output power-down control — This bit controls the output power-down of I/Os. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled 1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In STOP mode, only pad power sequence driver is disabled but the state of the output is kept. In STANDBY mode, power sequence driver and all pads except those mapped on wakeup lines are not powered and therefore high impedance. Wakeup line configuration remains unchanged.
MVRON	Main voltage regulator control — This bit specifies whether main voltage regulator is switched off or not while entering this mode. 0 Main voltage regulator is switched off 1 Main voltage regulator is switched on
DFLAON	Data flash power-down control — This bit specifies the operating mode of the data flash after entering this mode. 00 reserved 01 Data flash is in power-down mode 10 Data flash is in low-power mode 11 Data flash is in normal mode Note: If the flash memory is to be powered down in any mode, then your software must ensure that reset sources are configured as long resets in the RGM_FESS register (see <a href="#">Section 9.3.1.7: Functional Event Short Sequence Register (RGM_FESS)</a> ).

**Table 71. Mode Configuration Registers (ME\_<mode>\_MC) Field Descriptions (continued)**

Field	Description
CFLAON	Code flash power-down control — This bit specifies the operating mode of the program flash after entering this mode. 00 reserved 01 Code flash is in power-down mode 10 Code flash is in low-power mode 11 Code flash is in normal mode
FMPLLON	frequency modulated phase locked loop control 0 frequency modulated phase locked loop is switched off 1 frequency modulated phase locked loop is switched on
FXOSCON	fast external crystal oscillator (4-16 MHz) control 0 fast external crystal oscillator (4-16 MHz) is switched off 1 fast external crystal oscillator (4-16 MHz) is switched on
FIRCON	fast internal RC oscillator (16 MHz) control 0 fast internal RC oscillator (16 MHz) is switched off 1 fast internal RC oscillator (16 MHz) is switched on
SYSCLK	System clock switch control — These bits specify the system clock to be used by the system. 0000 16 MHz int. RC osc. 0001 div. 16 MHz int. RC osc. 0010 4-16 MHz ext. xtal osc. 0011 div. ext. xtal osc. 0100 freq. mod. PLL 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled

8.3.1.16 Peripheral Status Register 0 (ME\_PS0)

Address 0xC3FD\_C060 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	S_DMA_CH_MUX	0	S_FlexCAN5	S_FlexCAN4	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	S_LINFlex9	S_LINFlex8	0	0	S_DSP15	S_DSP14	S_DSP13	S_DSP12	S_DSP11	S_DSP10	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 65. Peripheral Status Register 0 (ME\_PS0)

This register provides the status of the peripherals. Please see [Table 72](#) for details.

8.3.1.17 Peripheral Status Register 1 (ME\_PS1)

Address 0xC3FD\_C064 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_CANSampler	0	0	S_CTU	0	S_LINFlex7	S_LINFlex6	S_LINFlex5	S_LINFlex4	S_LINFlex3	S_LINFlex2	S_LINFlex1	S_LINFlex0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	S_I2C0	0	0	0	0	0	0	0	0	0	0	S_ADC1	S_ADC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 66. Peripheral Status Register 1 (ME\_PS1)



This register provides the status of the peripherals. Please see [Table 72](#) for details.

**8.3.1.18 Peripheral Status Register 2 (ME\_PS2)**

Address 0xC3FD\_C068 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_PIT_RTI	S_RTC_API	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	S_eMIOS1	S_eMIOS0	0	0	S_WKPU	S_SIUL	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 67. Peripheral Status Register 2 (ME\_PS2)**

This register provides the status of the peripherals. Please see [Table 72](#) for details.

**8.3.1.19 Peripheral Status Register 3 (ME\_PS3)**

Address 0xC3FD\_C06C Access: User read, Supervisor read, Test read

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	S_CMU	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 68. Peripheral Status Register 3 (ME\_PS3)**

This register provides the status of the peripherals. Please see [Table 72](#) for details.

**Table 72. Peripheral Status Registers 0...4 (ME\_PS0...4) Field Descriptions**

Field	Description
S_<periph>	Peripheral status — These bits specify the current status of the peripherals in the system. If no peripheral is mapped on a particular position, the corresponding bit is always read as '0'. 0 Peripheral is frozen 1 Peripheral is active

**8.3.1.20 Run Peripheral Configuration Registers (ME\_RUN\_PC0...7)**

Address 0xC3FD\_C080 - 0xC3FD\_C09C

Access: User read, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 69. Run Peripheral Configuration Registers (ME\_RUN\_PC0...7)**

These registers configure eight different types of peripheral behavior during run modes.

**Table 73. Run Peripheral Configuration Registers (ME\_RUN\_PC0...7) Field Descriptions**

Field	Description
RUN3	Peripheral control during RUN3 0 Peripheral is frozen with clock gated 1 Peripheral is active
RUN2	Peripheral control during RUN2 0 Peripheral is frozen with clock gated 1 Peripheral is active
RUN1	Peripheral control during RUN1 0 Peripheral is frozen with clock gated 1 Peripheral is active
RUN0	Peripheral control during RUN0 0 Peripheral is frozen with clock gated 1 Peripheral is active
DRUN	Peripheral control during DRUN 0 Peripheral is frozen with clock gated 1 Peripheral is active

**Table 73. Run Peripheral Configuration Registers (ME\_RUN\_PC0...7)  
Field Descriptions (continued)**

Field	Description
SAFE	Peripheral control during SAFE 0 Peripheral is frozen with clock gated 1 Peripheral is active
TEST	Peripheral control during TEST 0 Peripheral is frozen with clock gated 1 Peripheral is active
RESET	Peripheral control during RESET 0 Peripheral is frozen with clock gated 1 Peripheral is active

**8.3.1.21 Low-Power Peripheral Configuration Registers (ME\_LP\_PC0...7)**

Address 0xC3FD\_C0A0 - 0xC3FD\_C0BC

Access: User read, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STANDBY	0	0	STOP	0	HALT	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 70. Low-Power Peripheral Configuration Registers (ME\_LP\_PC0...7)**

These registers configure eight different types of peripheral behavior during non-run modes.

**Table 74. Low-Power Peripheral Configuration Registers (ME\_LP\_PC0...7)  
Field Descriptions**

Field	Description
STANDBY	Peripheral control during STANDBY 0 Peripheral is frozen with clock gated 1 Peripheral is active
STOP	Peripheral control during STOP 0 Peripheral is frozen with clock gated 1 Peripheral is active
HALT	Peripheral control during HALT 0 Peripheral is frozen with clock gated 1 Peripheral is active



### 8.3.1.22 Peripheral Control Registers (ME\_PCTL0...143)

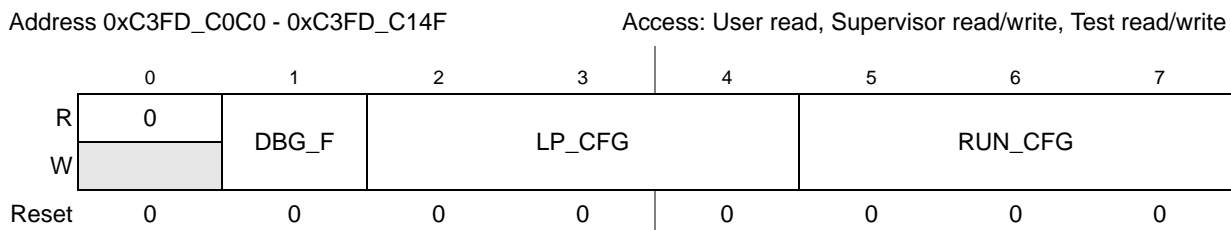


Figure 71. Peripheral Control Registers (ME\_PCTL0...143)

These registers select the configurations during run and non-run modes for each peripheral.

Table 75. Peripheral Control Registers (ME\_PCTL0...143) Field Descriptions

Field	Description
DBG_F	Peripheral control in debug mode — This bit controls the state of the peripheral in debug mode. 0 Peripheral state depends on RUN_CFG/LP_CFG bits and the device mode. 1 Peripheral is frozen if not already frozen in device modes. <b>Note:</b> This feature is useful to freeze the peripheral state while entering debug. For example, this may be used to prevent a reference timer from running while making a debug accesses.
LP_CFG	Peripheral configuration select for non-run modes — These bits associate a configuration as defined in the ME_LP_PC0...7 registers to the peripheral. 000 Selects ME_LP_PC0 configuration 001 Selects ME_LP_PC1 configuration 010 Selects ME_LP_PC2 configuration 011 Selects ME_LP_PC3 configuration 100 Selects ME_LP_PC4 configuration 101 Selects ME_LP_PC5 configuration 110 Selects ME_LP_PC6 configuration 111 Selects ME_LP_PC7 configuration
RUN_CFG	Peripheral configuration select for run modes — These bits associate a configuration as defined in the ME_RUN_PC0...7 registers to the peripheral. 000 Selects ME_RUN_PC0 configuration 001 Selects ME_RUN_PC1 configuration 010 Selects ME_RUN_PC2 configuration 011 Selects ME_RUN_PC3 configuration 100 Selects ME_RUN_PC4 configuration 101 Selects ME_RUN_PC5 configuration 110 Selects ME_RUN_PC6 configuration 111 Selects ME_RUN_PC7 configuration

## 8.4 Functional Description

### 8.4.1 Mode Transition Request

The transition from one mode to another mode is normally handled by software by accessing the mode control ME\_MCTL register. But in case of special events, mode

transition can be automatically managed by hardware. In order to switch from one mode to another, the application should access ME\_MCTL register twice by writing

- the first time with the value of the key (0x5AF0) into the KEY bit field and the required target mode into the TARGET\_MODE bit field,
- and the second time with the inverted value of the key (0xA50F) into the KEY bit field and the required target mode into the TARGET\_MODE bit field.

Once a valid mode transition request is detected, the target mode configuration information is loaded from the corresponding ME\_<mode>\_MC register. The mode transition request may require a number of cycles depending on the programmed configuration, and software should check the S\_CURRENT\_MODE bit field and the S\_MTRANS bit of the global status register ME\_GS to verify when the mode has been correctly entered and the transition process has completed. For a description of valid mode requests, please see [Section 8.4.5: Mode Transition Interrupts](#).

Any modification of the mode configuration register of the currently selected mode will not be taken into account immediately but on the next request to enter this mode. This means that transition requests such as RUN0...3 → RUN0...3, DRUN → DRUN, SAFE → SAFE, and TEST → TEST are considered valid mode transition requests. As soon as the mode request is accepted as valid, the S\_MTRANS bit is set till the status in the ME\_GS register matches the configuration programmed in the respective ME\_<mode>\_MC register.

*Note: It is recommended that software poll the S\_MTRANS bit in the ME\_GS register after requesting a transition to HALT, STOP, or STANDBY modes.*

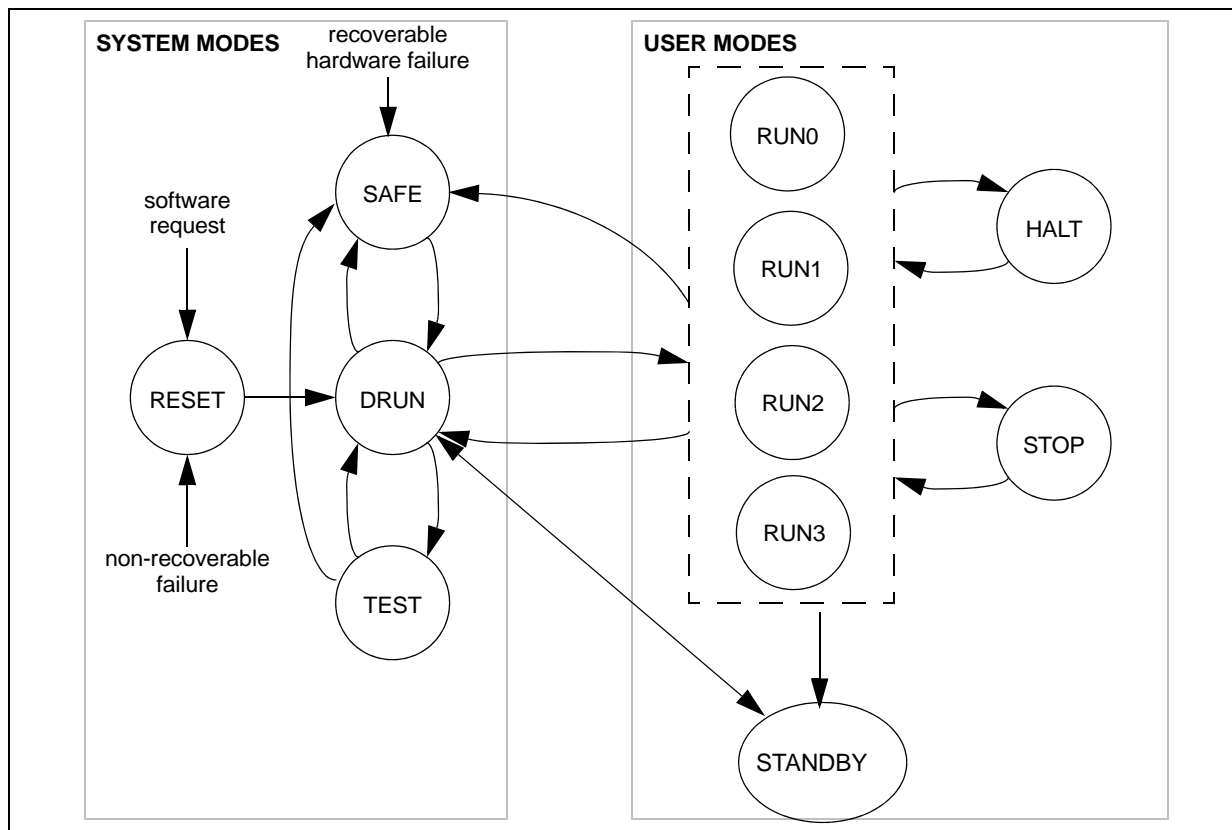


Figure 72. MC\_ME Mode Diagram

## 8.4.2 Modes Details

### 8.4.2.1 RESET Mode

The device enters this mode on the following events:

- from SAFE, DRUN, RUN0...3, or TEST mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with "0000"
- from any mode due to a system reset by the MC\_RGM because of some non-recoverable hardware failure in the system (see the MC\_RGM chapter for details)

Transition to this mode is instantaneous, and the system remains in this mode until the reset sequence is finished. The mode configuration information for this mode is provided by the ME\_RESET\_MC register. This mode has a pre-defined configuration, and the 16 MHz int. RC osc. is selected as the system clock. All power domains are made active in this mode.

### 8.4.2.2 DRUN Mode

The device enters this mode on the following events.

- automatically from RESET mode after completion of the reset sequence
- from RUN0...3, SAFE, or TEST mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with "0011"
- from the STANDBY mode after an external wakeup event or internal wakeup alarm (e.g. RTC/API event)

As soon as any of the above events has occurred, a DRUN mode transition request is generated. The mode configuration information for this mode is provided by the ME\_DRUN\_MC register. In this mode, the flashes, all clock sources, and the system clock configuration can be controlled by software as required. After system reset, the software execution starts with the default configuration selecting the 16 MHz int. RC osc. as the system clock.

This mode is intended to be used by software

- to initialize all registers as per the system needs
- to execute small routines in a 'ping-pong' with the STANDBY mode

When this mode is entered from STANDBY after a wakeup event, the ME\_DRUN\_MC register content is restored to its pre-STANDBY values, and the mode starts in that configuration.

All power domains are active when this mode is entered due to a system reset sequence initiated by a destructive reset event. In other cases of entry, such as the exit from STANDBY after a wakeup event, a functional reset event like an external reset or a software request from RUN0...3, SAFE, or TEST mode, active power domains are determined by the power configuration registers PCU\_PCONF2...3 of the MC\_PCU. All power domains except power domains #0 and #1 are configurable in this mode (see the MC\_PCU chapter for details).

*Note:* As flashes can be configured in low-power or power-down state in this mode, software must ensure that the code executes from RAM before changing to this mode.

### 8.4.2.3 SAFE Mode

The device enters this mode on the following events:

- from DRUN, RUN0...3, or TEST mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with "0010"
- from any mode except RESET due to a SAFE mode request generated by the MC\_RGM because of some potentially recoverable hardware failure in the system (see the MC\_RGM chapter for details)

As soon as any of the above events has occurred, a SAFE mode transition request is generated. The mode configuration information for this mode is provided by the ME\_SAFE\_MC register. This mode has a pre-defined configuration, and the 16 MHz int. RC osc. is selected as the system clock. All power domains are made active in this mode.

If the SAFE mode is requested by software while some other mode transition process is ongoing, the new target mode becomes the SAFE mode regardless of other pending requests. In this case, the new mode request is not interpreted as an invalid request.

*Note: If software requests to change to the SAFE mode and then requests to change back to the parent mode before the mode transition is completed, the device's final mode after mode transition will be the parent mode. However, this is not recommended software behavior. It is recommended for software to wait until the S\_MTRANS bit is cleared after requesting a change to SAFE before requesting another mode change.*

As long as a SAFE event is active, the system remains in the SAFE mode and no write access is allowed to the ME\_MCTL register.

This mode is intended to be used by software

- to assess the severity of the cause of failure and then to either
  - re-initialize the device via the DRUN mode, or
  - completely reset the device via the RESET mode.

If the outputs of the system I/Os need to be forced to a high impedance state upon entering this mode, the PDO bit of the ME\_SAFE\_MC register should be set. In this case, the pads' power sequence driver cell is also disabled. The input levels remain unchanged.

### 8.4.2.4 TEST Mode

The device enters this mode on the following events:

- from the DRUN mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with "0001"

As soon as any of the above events has occurred, a TEST mode transition request is generated. The mode configuration information for this mode is provided by the ME\_TEST\_MC register. Except for the main voltage regulator, all resources of the system are configurable in this mode. The system clock to the whole system can be stopped by programming the SYSCLK bit field to "1111", and in this case, the only way to exit this mode is via a device reset.

This mode is intended to be used by software

- to execute on-chip test routines

All power domains except power domains #0 and #1 are configurable in this mode. Active power domains are determined by the power configuration registers PCU\_PCONF2...3 of the MC\_PCU.

*Note:* As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from RAM before it changes to this mode.

#### 8.4.2.5 RUN0...3 Modes

The device enters one of these modes on the following events:

- from the DRUN another RUN0...3 mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with "0100...0111"
- from the HALT mode by an interrupt event
- from the STOP mode by an interrupt or wakeup event

As soon as any of the above events occur, a RUN0...3 mode transition request is generated. The mode configuration information for these modes is provided by ME\_RUN0...3\_MC registers. In these modes, the flashes, all clock sources, and the system clock configuration can be controlled by software as required.

These modes are intended to be used by software

- to execute application routines

All power domains except power domains #0 and #1 are configurable in these modes in order to reduce leakage consumption. Active power domains are determined by the power configuration registers PCU\_PCONF2...3 of the MC\_PCU.

*Note:* As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from RAM before it changes to this mode.

#### 8.4.2.6 HALT Mode

The device enters this mode on the following events:

- from one of the RUN0...3 modes when the TARGET\_MODE bit field of the ME\_MCTL register is written with "1000".

As soon as any of the above events occur, a HALT mode transition request is generated. The mode configuration information for this mode is provided by ME\_HALT\_MC register. This mode is quite configurable, and the ME\_HALT\_MC register should be programmed according to the system needs. The main voltage regulator and the flashes can be put in power-down mode as needed. If there is a HALT mode request while an interrupt request is active, the device mode does not change, and an invalid mode interrupt is not generated.

This mode is intended as a first level low-power mode with

- the core clock frozen
- only a few peripherals running

and to be used by software

- to wait until it is required to do something and then to react quickly (i.e. within a few system clock cycles of an interrupt event)

All power domains except power domains #0 and #1 are configurable in this mode in order to reduce leakage consumption. Active power domains are determined by the power configuration registers PCU\_PCONF2...3 of the MC\_PCU.

### 8.4.2.7 STOP Mode

The device enters this mode on the following events:

- from one of the RUN0...3 modes when the TARGET\_MODE bit field of the ME\_MCTL register is written with "1010".

As soon as any of the above events occur, a STOP mode transition request is generated. The mode configuration information for this mode is provided by the ME\_STOP\_MC register. This mode is fully configurable, and the ME\_STOP\_MC register should be programmed according to the system needs. The FMPLL is switched off in this mode. The main voltage regulator and the flashes can be put in power-down mode as needed. If there is a STOP mode request while any interrupt or wakeup event is active, the device mode does not change, and an invalid mode interrupt is not generated.

This can be used as an advanced low-power mode with the core clock frozen and almost all peripherals stopped.

This mode is intended as an advanced low-power mode with

- the core clock frozen
- almost all peripherals stopped

and to be used by software

- to wait until it is required to do something with no need to react quickly (e.g. allow for system clock source to be re-started)

If the pads' power sequence driver cell needs to be disabled while entering this mode, the PDO bit of the ME\_STOP\_MC register should be set. The state of the outputs is kept.

This mode can be used to stop all clock sources, thus preserving the device status. When exiting the STOP mode, the fast internal RC oscillator (16 MHz) clock is selected as the system clock until the target clock is available.

All power domains except power domains #0 and #1 are configurable in this mode in order to reduce leakage consumption. Active power domains are determined by the power configuration registers PCU\_PCONF2...3 of the MC\_PCU.

### 8.4.2.8 STANDBY Mode

The device enters this mode on the following events:

- from the DRUN or one of the RUN0...3 modes when the TARGET\_MODE bit field of the ME\_MCTL register is written with "1101".

As soon as any of the above events occur, a STANDBY mode transition request is generated. The mode configuration information for this mode is provided by the ME\_STANDBY\_MC register. In this mode, the power supply is turned off for most of the device. The only parts of the device that are still powered during this mode are pads mapped on wakeup lines and power domain #0 which contains the MC\_RGM, MC\_PCU, WKPU, 8K RAM (0x0000 - 0x1FFF), RTC\_API, CANSampler, SIRC, FIRC, SXOSC, and device and user option bits. The FIRC can be optionally switched off. This is the lowest power consumption mode possible on the device.

This mode is intended as an extreme low-power mode with

- the core, the flashes, and almost all peripherals and memories powered down

and to be used by software

- to wait until it is required to do something with no need to react quickly (i.e. allow for system power-up and system clock source to be re-started)

The exit sequence of this mode is similar to the reset sequence. However, in addition to booting from the default location, the device can also be configured to boot from the backup RAM (see the RGM\_STDBY register description in the MC\_RGM chapter for details). In the case of booting from backup RAM, it is also possible to keep the flashes disabled by writing "01" to the CFLAON and DFLAON fields in the ME\_DRUN\_MC register prior to STANDBY entry.

If there is a STANDBY mode request while any wakeup event is active, the device mode does not change.

All power domains except power domain #0 are configurable in this mode in order to reduce leakage consumption. Active power domains are determined by the power configuration registers PCU\_PCONF2...3 of the MC\_PCU.

### 8.4.3 Mode transition process

The process of mode transition follows the following steps in a pre-defined manner depending on the current device mode and the requested target mode. In many cases of mode transition, not all steps need to be executed based on the mode control information, and some steps may not be valid according to the mode definition itself.

#### 8.4.3.1 Target Mode Request

The target mode is requested by accessing the ME\_MCTL register with the required keys. This mode transition request by software must be a valid request satisfying a set of pre-defined rules to initiate the process. If the request fails to satisfy these rules, it is ignored, and the TARGET\_MODE bit field is not updated. An optional interrupt can be generated for invalid mode requests. See [Section 8.4.5: Mode Transition Interrupts](#) for details.

In the case of mode transitions occurring because of hardware events such as a reset, a SAFE mode request, or interrupt requests and wakeup events to exit from low-power modes, the TARGET\_MODE bit field of the ME\_MCTL register is automatically updated with the appropriate target mode. The mode change process start is indicated by the setting of the mode transition status bit S\_MTRANS of the ME\_GS register.

A RESET mode requested via the ME\_MCTL register is passed to the MC\_RGM, which generates a global system reset and initiates the reset sequence. The RESET mode request has the highest priority, and the MC\_ME is kept in the RESET mode during the entire reset sequence.

The SAFE mode request has the next highest priority after reset which can be generated by software via the ME\_MCTL register from all software running modes including DRUN, RUN0...3, and TEST or by the MC\_RGM after the detection of system hardware failures, which may occur in any mode.

#### 8.4.3.2 Target Mode Configuration Loading

On completion of the [Section 8.4.3.1: Target Mode Request](#), the target mode configuration from the ME\_<target mode>\_MC register is loaded to start the resources (voltage sources, clock sources, flashes, pads, etc.) control process.

An overview of resource control possibilities for each mode is shown in [Table 76](#). A ‘√’ indicates that a given resource is configurable for a given mode.

**Table 76. MC\_ME Resource Control Overview**

Resource	Mode							
	RESET	TEST	SAFE	DRUN	RUN0...3	HALT	STOP	STANDBY
FIRC	on	√ on	on	on	on	√ on	√ on	√ on
FXOSC	off	√ off	off	√ off	√ off	√ off	√ off	off
FMPLL	off	√ off	off	√ off	√ off	√ off	off	off
CFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down	power-down
DFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down	power-down
MVREG	on	on	on	on	on	√ on	√ on	off
PDO	off	√ off	√ on	off	off	off	√ off	on

**8.4.3.3 Peripheral Clocks Disable**

On completion of the [Section 8.4.3.1: Target Mode Request](#), the MC\_ME requests each peripheral to enter its stop mode when:

- the peripheral is configured to be disabled via the target mode, the peripheral configuration registers ME\_RUN\_PC0...7 and ME\_LP\_PC0...7, and the peripheral control registers ME\_PCTL0...143

**Caution:** The MC\_ME does not automatically request peripherals to enter their stop modes if the power domains in which they are residing are to be turned off due to a mode change. Therefore, it is software’s responsibility to ensure that those peripherals that are to be powered down are configured in the MC\_ME to be frozen.

Each peripheral acknowledges its stop mode request after closing its internal activity. The MC\_ME then disables the corresponding clock(s) to this peripheral.

In the case of a SAFE mode transition request, the MC\_ME does not wait for the peripherals to acknowledge the stop requests. The SAFE mode clock gating configuration is applied immediately regardless of the status of the peripherals’ stop acknowledges.

Please see [Section 8.4.6: Peripheral Clock Gating](#), for more details.





Each peripheral that may block or disrupt a communication bus to which it is connected ensures that these outputs are forced to a safe or recessive state when the device enters the SAFE mode.

#### 8.4.3.4 Processor Low-Power Mode Entry

If, on completion of the [Section 8.4.3.3: Peripheral Clocks Disable](#), the mode transition is to the HALT mode, the MC\_ME requests the processor to enter its halted state. The processor acknowledges its halt state request after completing all outstanding bus transactions.

If, on completion of the [Section 8.4.3.3: Peripheral Clocks Disable](#), the mode transition is to the STOP or STANDBY mode, the MC\_ME requests the processor to enter its stopped state. The processor acknowledges its stop state request after completing all outstanding bus transactions.

#### 8.4.3.5 Processor and System Memory Clock Disable

If, on completion of the [Section 8.4.3.14: Processor Low-Power Mode Exit](#), the mode transition is to the HALT, STOP, or STANDBY mode and the processor is in its appropriate halted or stopped state, the MC\_ME disables the processor and system memory clocks to achieve further power saving.

The clocks to the processor and system memories are unaffected for all transitions between software running modes including DRUN, RUN0...3, and SAFE.

**Caution:** Clocks to the whole device including the processor and system memories can be disabled in TEST mode.

#### 8.4.3.6 Clock sources switch-on

On completion of the [Section 8.4.3.4: Processor Low-Power Mode Entry](#), the MC\_ME controls all clock sources that affect the system clock based on the <clock source>ON bits of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers. The following system clock sources are controlled at this step:

- the fast internal RC oscillator (16 MHz)
- the fast external crystal oscillator (4-16 MHz)

*Note:* The frequency modulated phase locked loop, which needs the main voltage regulator to be stable, is not controlled by this step.

The clock sources that are required by the target mode are switched on. The duration required for the output clocks to be stable depends on the type of source, and all further steps of mode transition depending on one or more of these clocks waits for the stable status of the respective clocks. The availability status of these system clocks is updated in the S\_<clock source> bits of ME\_GS register.

The clock sources which need to be switched off are unaffected during this process in order to not disturb the system clock which might require one of these clocks before switching to a different target clock.

It is also possible to automatically switch-on the FXOSC after exiting STANDBY mode, by programming ME\_DRUN\_MC[FXOSC] = 1 prior to STANDBY entry. After the chip exits STANDBY mode, your software should wait for FXOSC to be switched-on before any mode transition request in order to avoid a Mode Request Illegal Event.

### 8.4.3.7 Main Voltage Regulator Switch-On

On completion of the [Section 8.4.3.1: Target Mode Request](#), if the main voltage regulator needs to be switched on from its off state based on the MVRON bit of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, the MC\_ME requests the MC\_PCU to power-up the regulator and waits for the output voltage stable status in order to update the S\_MVR bit of the ME\_GS register.

This step is required only during the exit of the low-power modes HALT and STOP. In this step, the fast internal RC oscillator (16 MHz) is switched on regardless of the target mode configuration, as the main voltage regulator requires the 16 MHz int. RC osc. during power-up in order to generate the voltage status.

During the STANDBY exit sequence, the MC\_PCU alone manages the power-up of the main voltage regulator, and the MC\_ME is kept in RESET or shut off (depending on the power domain #1 status).

### 8.4.3.8 Flash Modules Switch-On

On completion of the [Section 8.4.3.7: Main Voltage Regulator Switch-On](#), if a flash module needs to be switched to normal mode from its low-power or power-down mode based on the CFLAON and DFLAON bit fields of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, the MC\_ME requests the flash to exit from its low-power/power-down mode. When the flash modules are available for access, the S\_CFLA and S\_DFLA bit fields of the ME\_GS register are updated to "11" by hardware.

If the main regulator is also off in device low-power modes, then during the exit sequence, the flash is kept in its low-power state and is switched on only when the [Section 8.4.3.7: Main Voltage Regulator Switch-On](#) process has completed.

**Caution:** It is illegal to switch the flashes from low-power mode to power-down mode and from power-down mode to low-power mode. The MC\_ME, however, does not prevent this nor does it flag it.

### 8.4.3.9 FMPLL Switch-On

On completion of the [Section 8.4.3.6: Clock sources switch-on](#) and [Section 8.4.3.7: Main Voltage Regulator Switch-On](#), if the FMPLL is to be switched on from the off state based on the FMPLLON bit of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, the MC\_ME requests the FMPLL digital interface to start the phase locking process and waits for the FMPLL to enter into the locked state. When the FMPLL enters the locked state and starts providing a stable output clock, the S\_FMPLL bit of ME\_GS register is set.

### 8.4.3.10 Power Domain #2...3 Switch-On

On completion of the [Section 8.4.3.7: Main Voltage Regulator Switch-On](#), the MC\_ME indicates a mode change to the MC\_PCU. The MC\_PCU then determines whether a power-up sequence is required for power domains #2 and #3. Only after the MC\_PCU has executed all required power-ups does the MC\_ME complete the mode transition.

#### 8.4.3.11 Pad Outputs-On

On completion of the [Section 8.4.3.7: Main Voltage Regulator Switch-On](#), if the PDO bit of the ME\_<target mode>\_MC register is cleared, then

- all pad outputs are enabled to return to their previous state
- the I/O pads power sequence driver is switched on

#### 8.4.3.12 Peripheral Clocks Enable

Based on the current and target device modes, the peripheral configuration registers ME\_RUN\_PC0...7, ME\_LP\_PC0...7, and the peripheral control registers ME\_PCTL0...143, the MC\_ME enables the clocks for selected modules as required. This step is executed only after the [Section 8.4.3.7: Main Voltage Regulator Switch-On](#) process is completed.

Also if a mode change translates to a power up of one or more power domains, the MC\_PCU indicates the MC\_ME after completing the power-up sequence upon which the MC\_ME may assert the peripheral clock enables of the peripherals residing in those power domains.

#### 8.4.3.13 Processor and Memory Clock Enable

If the mode transition is from any of the low-power modes HALT or STOP to RUN0...3, the clocks to the processor and system memories are enabled. The process of enabling these clocks is executed only after the [Section 8.4.3.8: Flash Modules Switch-On](#) process is completed.

#### 8.4.3.14 Processor Low-Power Mode Exit

If the mode transition is from any of the low-power modes HALT, STOP, or STANDBY to RUN0...3, the MC\_ME requests the processor to exit from its halted or stopped state. This step is executed only after the [Section 8.4.3.13: Processor and Memory Clock Enable](#) process is completed.

### 8.4.3.15 System Clock Switching

Based on the SYSCLK bit field of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, if the target and current system clock configurations differ, the following method is implemented for clock switching.

- The target clock configuration for the 16 MHz int. RC osc. is effective only when the S\_FIRC bit of the ME\_GS register is set by hardware (i.e. the fast internal RC oscillator (16 MHz) has stabilized).
- The target clock configuration for the div. 16 MHz int. RC osc. is effective only when the S\_FIRC bit of the ME\_GS register is set by hardware (i.e. the fast internal RC oscillator (16 MHz) has stabilized).
- The target clock configuration for the 4-16 MHz ext. xtal osc. is effective only when the S\_FXOSC bit of the ME\_GS register is set by hardware (i.e. the fast external crystal oscillator (4-16 MHz) has stabilized).
- The target clock configuration for the div. ext. xtal osc. is effective only when the S\_FXOSC bit of the ME\_GS register is set by hardware (i.e. the fast external crystal oscillator (4-16 MHz) has stabilized).
- The target clock configuration for the freq. mod. PLL is effective only when the S\_FMPLL bit of the ME\_GS register is set by hardware (i.e. the frequency modulated phase locked loop has stabilized).
- If the clock is to be disabled, the SYSCLK bit field should be programmed with "1111". This is possible only in the STOP and TEST modes. In the STANDBY mode, the clock configuration is fixed, and the system clock is automatically forced to '0'.

The current system clock configuration can be observed by reading the S\_SYSCLK bit field of the ME\_GS register, which is updated after every system clock switching. Until the target clock is available, the system uses the previous clock configuration.

System clock switching starts only after

- the [Section 8.4.3.6: Clock sources switch-on](#) process has completed if the target system clock source needs to be switched on
- the [Section 8.4.3.9: FMPLL Switch-On](#) process has completed if the target system clock is the *freq. mod. PLL*
- the [Section 8.4.3.3: Peripheral Clocks Disable](#) process is completed in order not to change the system clock frequency before peripherals close their internal activities

An overview of system clock source selection possibilities for each mode is shown in [Table 77](#). A '√' indicates that a given clock source is selectable for a given mode.

Table 77. MC\_ME System Clock Selection Overview

System Clock Source	Mode							
	RESET	TEST	SAFE	DRUN	RUN0...3	HALT	STOP	STANDBY
16 MHz int. RC osc.	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	
div. 16 MHz int. RC osc.		√		√	√	√	√	
4-16 MHz ext. xtal osc.		√		√	√	√	√	
div. ext. xtal osc.		√		√	√	√	√	
freq. mod. PLL		√		√	√	√		
system clock is disabled		√ <sup>(1)</sup>					√	√ (default)

1. Disabling the system clock during TEST mode will require a reset in order to exit TEST mode.

**8.4.3.16 Power Domain #2...3 Switch-Off**

Based on the device mode and the MC\_PCU's power configuration registers PCU\_PCONF2...3, the power domains #2 to 3 are controlled by the MC\_PCU.

If a mode change translates to a power-down of a power domain, then the MC\_PCU starts the power-down sequence. The MC\_PCU acknowledges the completion of the power-down sequences with respect to the new mode, and the MC\_ME uses this information to update the mode transition status. This step is executed only after the [Section 8.4.3.3: Peripheral Clocks Disable](#) process has completed.

**8.4.3.17 Pad Switch-Off**

If the PDO bit of the ME\_<target mode>\_MC register is '1' then

- the outputs of the pads are forced to the high impedance state if the target mode is SAFE or TEST
- I/O pads power sequence driver is switched off if the target mode is one of SAFE, TEST, or STOP modes

In STANDBY mode, the power sequence driver and all pads except the external reset and those mapped on wakeup lines are not powered and therefore high impedance. The wakeup line configuration remains unchanged.

This step is executed only after the [Section 8.4.3.3: Peripheral Clocks Disable](#) process is completed.

### 8.4.3.18 FMPLL Switch-Off

Based on the FMPLLON bit of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, if FMPLL is to be switched off, the MC\_ME requests the FMPLL to power down and updates its availability status bit S\_FMPLL of the ME\_GS register to '0'. This step is executed only after the [Section 8.4.3.15: System Clock Switching](#) process is completed.

### 8.4.3.19 Clock Sources Switch-Off

Based on the device mode and the <clock source>ON bits of the ME\_<mode>\_MC registers, if a given clock source is to be switched off, the MC\_ME requests the clock source to power down and updates its availability status bit S\_<clock source> of the ME\_GS register to '0'.

This step is executed only after

- [Section 8.4.3.15: System Clock Switching](#) process is completed in order not to lose the current system clock during mode transition.
- [Section 8.4.3.18: FMPLL Switch-Off](#) as the input reference clock of the FMPLL can be among these clock sources. This is needed to prevent an unwanted lock transition when the FMPLL is switched on.

### 8.4.3.20 Flash Switch-Off

Based on the CFLAON and DFLAON bit fields of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, if any of the flash modules is to be put in a low-power state, the MC\_ME requests the flash to enter the corresponding low-power state and waits for the deassertion of flash ready status signal. The exact low-power mode status of the flash modules is updated in the S\_CFLA and S\_DFLA bit fields of the ME\_GS register. This step is executed only when [Section 8.4.3.5: Processor and System Memory Clock Disable](#) process is completed.

### 8.4.3.21 Main Voltage Regulator Switch-Off

Based on the MVRON bit of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, if the main voltage regulator is to be switched off, the MC\_ME requests it to power down and clears the availability status bit S\_MVR of the ME\_GS register.

This step is required only during the entry of low-power modes like HALT and STOP. This step is executed only after completing the following processes:

- [Section 8.4.3.18: FMPLL Switch-Off](#)
- [Section 8.4.3.20: Flash Switch-Off](#)
- [Section 8.4.3.16: Power Domain #2...3 Switch-Off](#)
- [Section 8.4.3.10: Power Domain #2...3 Switch-On](#)
- the device consumption is less than the pre-defined threshold value (i.e. the S\_DC bit of the ME\_GS register is '0').

If the target mode is STANDBY, the main voltage regulator is not switched off by the MC\_ME and the STANDBY request is asserted after the above processes have completed upon which the MC\_PCU takes control of the main regulator. As the MC\_PCU needs the 16 MHz int. RC osc., the fast internal RC oscillator (16 MHz) remains active until all the STANDBY steps are executed by the MC\_PCU after which it may be switched off depending on the FIRCON bit of the ME\_STANDBY\_MC register.

### 8.4.3.22 Current Mode Update

The current mode status bit field S\_CURRENT\_MODE of the ME\_GS register is updated with the target mode bit field TARGET\_MODE of the ME\_MCTL register when:

- all the updated status bits in the ME\_GS register match the configuration specified in the ME\_<target mode>\_MC register
- power sequences are done
- clock disable/enable process is finished
- processor low-power mode (halt/stop) entry and exit processes are finished

Software can monitor the mode transition status by reading the S\_MTRANS bit of the ME\_GS register. The mode transition latency can differ from one mode to another depending on the resources' availability before the new mode request and the target mode's requirements.

### 8.4.4 Protection of Mode Configuration Registers

While programming the mode configuration registers ME\_<mode>\_MC, the following rules must be respected. Otherwise, the write operation is ignored and an invalid mode configuration interrupt may be generated.

- FIRC must be on if the system clock is one of the following:
  - 16 MHz int. RC osc.
  - div. 16 MHz int. RC osc.
- FXOSC must be on if the system clock is one of the following:
  - 4-16 MHz ext. xtal osc.
  - div. ext. xtal osc.

*Note:* Software must ensure to switch on the clock source that provides the input reference clock to the FMPLL. There is no automatic protection mechanism to check this in the MC\_ME.

- FMPLL must be on if the system clock is the freq. mod. PLL.
- Configuration "00" for the CFLAON and DFLAON bit fields are reserved.
- MVREG must be on if any of the following is active:
  - FMPLL
  - CFLASH
  - DFLASH
- System clock configurations marked as 'reserved' may not be selected.
- Configuration "1111" for the SYSCLK bit field is allowed only for the STOP and TEST modes, and only in this case may all system clock sources be turned off.

**Caution:** If the system clock is stopped during TEST mode, the device can exit only via a system reset.

### 8.4.5 Mode Transition Interrupts

The following are the three interrupts related to mode transition implemented in the MC\_ME.

#### 8.4.5.1 Invalid Mode Configuration Interrupt

Whenever a write operation is attempted to the ME\_<mode>\_MC registers violating the protection rules mentioned in the [Section 8.4.4: Protection of Mode Configuration Registers](#),

the interrupt pending bit I\_ICONF of the ME\_IS register is set and an interrupt request is generated if the mask bit M\_ICONF of ME\_IM register is '1'.

#### 8.4.5.2 Invalid Mode Transition Interrupt

The mode transition request is considered invalid under the following conditions:

- If the system is in the SAFE mode and the SAFE mode request from MC\_RGM is active, and if the target mode requested is other than RESET or SAFE, then this new mode request is considered to be invalid, and the S\_SEA bit of the ME\_IMTS register is set.
- If the TARGET\_MODE bit field of the ME\_MCTL register is written with a value different from the specified mode values (i.e. a non existing mode), an invalid mode transition event is generated. When such a non existing mode is requested, the S\_NMA bit of the ME\_IMTS register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME\_MCTL register.
- If some of the device modes are disabled as programmed in the ME\_ME register, their respective configurations are considered reserved, and any access to the ME\_MCTL register with those values results in an invalid mode transition request. When such a disabled mode is requested, the S\_DMA bit of the ME\_IMTS register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME\_MCTL register.
- If the target mode is not a valid mode with respect to current mode, the mode request illegal status bit S\_MRI of the ME\_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME\_MCTL register. Otherwise, the write operation is ignored.
- If further new mode requests occur while a mode transition is in progress (the S\_MTRANS bit of the ME\_GS register is '1'), the mode transition illegal status bit S\_MTI of the ME\_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME\_MCTL register. Otherwise, the write operation is ignored.

*Note:* As the causes of invalid mode transitions may overlap at the same time, the priority implemented for invalid mode transition status bits of the ME\_IMTS register in the order from highest to lowest is S\_SEA, S\_NMA, S\_DMA, S\_MRI, and S\_MTI.

As an exception, the mode transition request is not considered as invalid under the following conditions:

- A new request is allowed to enter the RESET or SAFE mode irrespective of the mode transition status.
- As the exit of HALT and STOP modes depends on the interrupts of the system which can occur at any instant, these requests to return to RUN0...3 modes are always valid.
- In order to avoid any unwanted lockup of the device modes, software can abort a mode transition by requesting the parent mode if, for example, the mode transition has not completed after a software determined 'reasonable' amount of time for whatever reason. The parent mode is the device mode before a valid mode request was made.
- Self-transition requests (e.g. RUN0 → RUN0) are not considered as invalid even when the mode transition process is active (i.e. S\_MTRANS is '1'). During the low-power mode exit process, if the system is not able to enter the respective RUN0...3 mode properly (i.e. all status bits of the ME\_GS register match with configuration bits in the ME\_<mode>\_MC register), then software can only request the SAFE or RESET mode. It is not possible to request any other mode or to go back to the low-power mode again.



Whenever an invalid mode request is detected, the interrupt pending bit I\_IMODE of the ME\_IS register is set, and an interrupt request is generated if the mask bit M\_IMODE is ME\_IM register is '1'.

#### 8.4.5.3 SAFE Mode Transition Interrupt

Whenever the system enters the SAFE mode as a result of a SAFE mode request from the MC\_RGM due to a hardware failure, the interrupt pending bit I\_SAFE of the ME\_IS register is set, and an interrupt is generated if the mask bit M\_SAFE of ME\_IM register is '1'.

The SAFE mode interrupt pending bit can be cleared only when the SAFE mode request is deasserted by the MC\_RGM (see the MC\_RGM chapter for details on how to clear a SAFE mode request). If the system is already in SAFE mode, any new SAFE mode request by the MC\_RGM also sets the interrupt pending bit I\_SAFE. However, the SAFE mode interrupt pending bit is not set when the SAFE mode is entered by a software request (i.e. programming of ME\_MCTL register).

#### 8.4.5.4 Mode Transition Complete interrupt

Whenever the system completes a mode transition fully (i.e. the S\_MTRANS bit of ME\_GS register transits from '1' to '0'), the interrupt pending bit I\_MTC of the ME\_IS register is set, and interrupt request is generated if the mask bit M\_MTC of the ME\_IM register is '1'. The interrupt bit I\_MTC is not set when entering low-power modes HALT and STOP in order to avoid the same event requesting the exit of these low-power modes.

### 8.4.6 Peripheral Clock Gating

During all device modes, each peripheral can be associated with a particular clock gating policy determined by two groups of peripheral configuration registers.

The run peripheral configuration registers ME\_RUN\_PC0...7 are chosen only during the software running modes DRUN, TEST, SAFE, and RUN0...3. All configurations are programmable by software according to the needs of application. Each configuration register contains a mode bit which determines whether or not a peripheral clock is to be gated. Run configuration selection for each peripheral is done by the RUN\_CFG bit field of the ME\_PCTL0...143 registers.

The low-power peripheral configuration registers ME\_LP\_PC0...7 are chosen only during the low-power modes HALT, STOP, and STANDBY. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit which determines whether or not a peripheral clock is to be gated. Low-power configuration selection for each peripheral is done by the LP\_CFG bit field of the ME\_PCTL0...143 registers.

Any modifications to the ME\_RUN\_PC0...7, ME\_LP\_PC0...7, and ME\_PCTL0...143 registers do not affect the clock gating behavior until a new mode transition request is generated.

Whenever the processor enters a debug session during any mode, the following occurs for each peripheral:

- The clock is gated if the DBG\_F bit of the associated ME\_PCTL0...143 register is set. Otherwise, the peripheral clock gating status depends on the RUN\_CFG and LP\_CFG bits. Any further modifications of the ME\_RUN\_PC0...7, ME\_LP\_PC0...7, and ME\_PCTL0...143 registers during a debug session will take affect immediately without requiring any new mode request.

### 8.4.7 Application Example

*Figure 73* shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

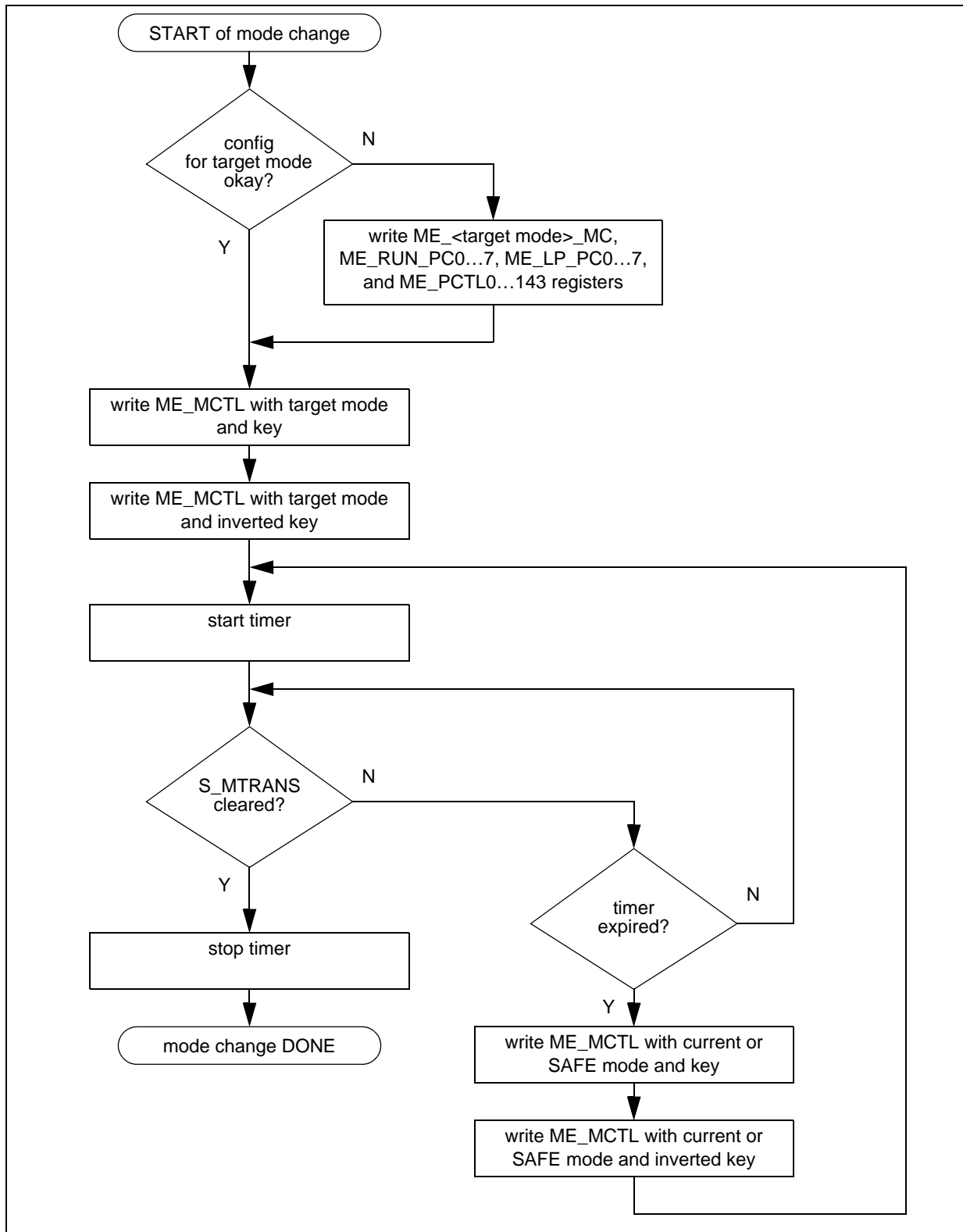


Figure 73. MC\_ME Application Example Flow Diagram

## 9 Reset Generation Module (MC\_RGM)

### 9.1 Introduction

#### 9.1.1 Overview

The reset generation module (MC\_RGM) centralizes the different reset sources and manages the reset sequence of the device. It provides a register interface and the reset sequencer. The different registers are available to monitor and control the device reset sequence. The reset sequencer is a state machine which controls the different phases (PHASE0, PHASE1, PHASE2, PHASE3, and IDLE) of the reset sequence and control the reset signals generated in the system.

*Figure 74* depicts the MC\_RGM block diagram.

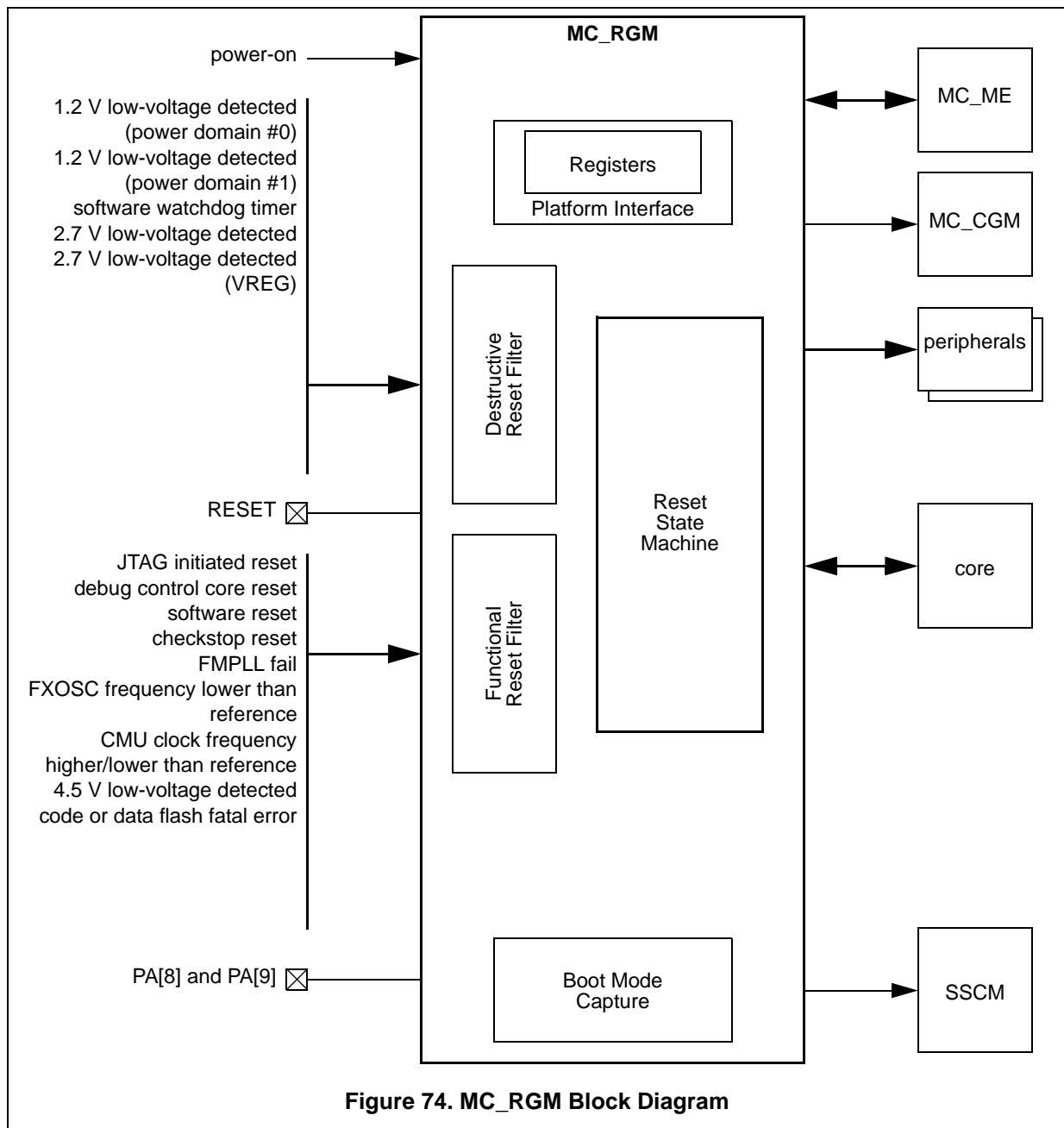


Figure 74. MC\_RGM Block Diagram

### 9.1.2 Features

The MC\_RGM contains the functionality for the following features:

- 'destructive' resets management
- 'functional' resets management
- signalling of reset events after each reset sequence (reset status flags)
- conversion of reset events to SAFE mode or interrupt request events (for further mode details, please see the MC\_ME chapter)
- short reset sequence configuration
- bidirectional reset behavior configuration
- selection of alternate boot via the backup RAM on STANDBY mode exit (for further mode details, please see the MC\_ME chapter)
- boot mode capture on RESET deassertion

### 9.1.3 Modes of operation

The different reset sources are organized into two families: 'destructive' and 'functional'.

- A 'destructive' reset source is associated with an event related to a critical - usually hardware - error or dysfunction. When a 'destructive' reset event occurs, the full reset sequence is applied to the device starting from PHASE0. This resets the full device ensuring a safe start-up state for both digital and analog modules. 'Destructive' resets are
  - power-on reset
  - 1.2 V low-voltage detected (power domain #0)
  - 1.2 V low-voltage detected (power domain #1)
  - software watchdog timer
  - 2.7 V low-voltage detected
  - 2.7 V low-voltage detected (VREG)
- A 'functional' reset source is associated with an event related to a less-critical - usually non-hardware - error or dysfunction. When a 'functional' reset event occurs, a partial reset sequence is applied to the device starting from PHASE1. In this case, most digital modules are reset normally, while analog modules or specific digital modules' (e.g. debug modules, flash modules) state is preserved. 'Functional' resets are
  - external reset
  - JTAG initiated reset
  - debug control core reset
  - software reset
  - checkstop reset
  - FMPLL fail
  - FXOSC frequency lower than reference
  - CMU clock frequency higher/lower than reference
  - 4.5 V low-voltage detected
  - code or data flash fatal error

When a reset is triggered, the MC\_RGM state machine is activated and proceeds through the different phases (i.e. PHASEn states). Each phase is associated with a particular device reset being provided to the system. A phase is completed when all corresponding phase

completion gates from either the system or internal to the MC\_RGM are acknowledged. The device reset associated with the phase is then released, and the state machine proceeds to the next phase up to entering the IDLE phase. During this entire process, the MC\_ME state machine is held in RESET mode. Only at the end of the reset sequence, when the IDLE phase is reached, does the MC\_ME enter the DRUN mode.

Alternatively, it is possible for software to configure some reset source events to be converted from a reset to either a SAFE mode request issued to the MC\_ME or to an interrupt issued to the core (see [Section 9.3.1.4: Destructive Event Reset Disable Register \(RGM\\_DERD\)](#) and [Section 9.3.1.6: Destructive Event Alternate Request Register \(RGM\\_DEAR\)](#) for 'destructive' resets and [Section 9.3.1.3: Functional Event Reset Disable Register \(RGM\\_FERD\)](#) and [Section 9.3.1.5: Functional Event Alternate Request Register \(RGM\\_FEAR\)](#) for 'functional' resets).

## 9.2 External Signal Description

The MC\_RGM interfaces to the bidirectional reset pin RESET and the boot mode pins PA[8] and PA[9].

## 9.3 Memory Map and Register Definition

Table 78. MC\_RGM Register Description

Address	Name	Description	Size	Access			Location
				Normal	Supervisor	Test	
0xC3FE_4000	RGM_FES	Functional Event Status	half-word	read	read/write <sup>(1)</sup>	read/write <sup>(1)</sup>	<a href="#">on page 234</a>
0xC3FE_4002	RGM_DES	Destructive Event Status	half-word	read	read/write <sup>(1)</sup>	read/write <sup>(1)</sup>	<a href="#">on page 235</a>
0xC3FE_4004	RGM_FERD	Functional Event Reset Disable	half-word	read	read/write <sup>(2)</sup>	read/write <sup>(2)</sup>	<a href="#">on page 236</a>
0xC3FE_4006	RGM_DERD	Destructive Event Reset Disable	half-word	read	read	read	<a href="#">on page 238</a>
0xC3FE_4010	RGM_FEAR	Functional Event Alternate Request	half-word	read	read/write	read/write	<a href="#">on page 239</a>
0xC3FE_4012	RGM_DEAR	Destructive Event Alternate Request	half-word	read	read	read	<a href="#">on page 240</a>
0xC3FE_4018	RGM_FESS	Functional Event Short Sequence	half-word	read	read/write	read/write	<a href="#">on page 241</a>
0xC3FE_401A	RGM_STDBY	STANDBY Reset Sequence	half-word	read	read/write	read/write	<a href="#">on page 243</a>
0xC3FE_401C	RGM_FBRE	Functional Bidirectional Reset Enable	half-word	read	read/write	read/write	<a href="#">on page 243</a>

1. individual bits cleared on writing '1'.
2. write once: '0' = disable, '1' = enable.

Note: Any access to unused registers as well as write accesses to read-only registers will:  
 not change register content  
 cause a transfer error

Table 79. MC\_RGM Memory Map

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0xC3FE_4000	RGM_FES / RGM_DES	R	F_EXR	0	0	0	0	0	0	0	F_FLASH	F_LVD45	F_CMU_FHL	F_CMU_OLR	F_FMPLL	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
		W	w1c								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
		R	F_POR	0	0	0	0	0	0	0	0	0	0	0	F_LVD27_VREG	F_LVD27	F_SWT	F_LVD12_PD1	F_LVD12_PD0
		W	w1c												w1c	w1c	w1c	w1c	w1c
0xC3FE_4004	RGM_FERD / RGM_DERD	R	D_EXR	0	0	0	0	0	0	0	D_FLASH	D_LVD45	D_CMU_FHL	D_CMU_OLR	D_FMPLL	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
		W																	
		R	0	0	0	0	0	0	0	0	0	0	0	0	D_LVD27_VREG	D_LVD27	D_SWT	D_LVD12_PD1	D_LVD12_PD0
		W																	
0xC3FE_4008 ... 0xC3FE_400C	reserved																		





Table 79. MC\_RGM Memory Map (continued)

Address	Name	0		1		2		3		27		5		6		7		8		9		10		11		12		13		14		15	
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																
0xC3FE_4010	RGM_FEAR/ RGM_DEAR	R	AR_EXR	0	0	0	0	0	0	0	0	0	0	0	0	0	AR_FLASH	AR_LVD45	AR_CMU_FHL	AR_CMU_OLR	AR_FMPPLL	AR_CHKSTOP	AR_SOFT	AR_CORE	AR_JTAG								
		W																															
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	AR_LVD27_VREG	AR_LVD27	AR_SWT	AR_LVD12_PD1	AR_LVD12_PD0							
		W																															
0xC3FE_4014	reserved																																
0xC3FE_4018	RGM_FESS/ RGM_STDBY	R	SS_EXR	0	0	0	0	0	0	0	0	0	0	0	0	0	SS_FLASH	SS_LVD45	SS_CMU_FHL	SS_CMU_OLR	SS_FMPPLL	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG								
		W																															
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BOOT_FROM_BKP_RAM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																															
0xC3FE_401C	RGM_FBRE	R	BE_EXR	0	0	0	0	0	0	0	0	0	0	0	0	0	BE_FLASH	BE_LVD45	BE_CMU_FHL	BE_CMU_OLR	BE_FMPPLL	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG								
		W																															
0xC3FE_4020 ... 0xC3FE_7FFC	reserved																																

### 9.3.1 Register descriptions

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the RGM\_STDBY register may be accessed as a word at address 0xC3FE\_4018, as a half-word at address 0xC3FE\_401A, or as a byte at address 0xC3FE\_401B.

#### 9.3.1.1 Functional Event Status Register (RGM\_FES)

Address 0xC3FE\_4000 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_EXR	0	0	0	0	0	0	F_FLASH	F_LVD45	F_CMU_FHL	F_CMU_OLR	F_FMPLL	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
W	w1c							w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 75. Functional Event Status Register (RGM\_FES)

This register contains the status of the last asserted functional reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared on write '1'.

Table 80. Functional Event Status Register (RGM\_FES) field descriptions

Field	Description
F_EXR	Flag for External Reset 0 No external reset event has occurred since either the last clear or the last destructive reset assertion 1 An external reset event has occurred
F_FLASH	Flag for code or data flash fatal error 0 No code or data flash fatal error event has occurred since either the last clear or the last destructive reset assertion 1 A code or data flash fatal error event has occurred
F_LVD45	Flag for 4.5 V low-voltage detected 0 No 4.5 V low-voltage detected event has occurred since either the last clear or the last destructive reset assertion 1 A 4.5 V low-voltage detected event has occurred
F_CMU_FHL	Flag for CMU clock frequency higher/lower than reference 0 No CMU clock frequency higher/lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A CMU clock frequency higher/lower than reference event has occurred
F_CMU_OLR	Flag for FXOSC frequency lower than reference 0 No FXOSC frequency lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A FXOSC frequency lower than reference event has occurred

**Table 80. Functional Event Status Register (RGM\_FES) field descriptions (continued)**

Field	Description
F_FMPLL	Flag for FMPLL fail 0 No FMPLL fail event has occurred since either the last clear or the last destructive reset assertion 1 A FMPLL fail event has occurred
F_CHKSTOP	Flag for checkstop reset 0 No checkstop reset event has occurred since either the last clear or the last destructive reset assertion 1 A checkstop reset event has occurred
F_SOFT	Flag for software reset 0 No software reset event has occurred since either the last clear or the last destructive reset assertion 1 A software reset event has occurred
F_CORE	Flag for debug control core reset 0 No debug control core reset event has occurred since either the last clear or the last destructive reset assertion 1 A debug control core reset event has occurred; this event can only be asserted when the DBCR0[RST] field is set by an external debugger. See the "Debug Support" chapter of the core reference manual for more details.
F_JTAG	Flag for JTAG initiated reset 0 No JTAG initiated reset event has occurred since either the last clear or the last destructive reset assertion 1 A JTAG initiated reset event has occurred

**9.3.1.2 Destructive Event Status Register (RGM\_DES)**

Address 0xC3FE\_4002

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_POR	0	0	0	0	0	0	0	0	0	0	F_LVD27_VREG	F_LVD27	F_SWT	F_LVD12_PD1	F_LVD12_PD0
W	w1c											w1c	w1c	w1c	w1c	w1c
POR	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 76. Destructive Event Status Register (RGM\_DES)**

This register contains the status of the last asserted destructive reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared on write '1'.

Table 81. Destructive Event Status Register (RGM\_DES) field descriptions

Field	Description
F_POR	Flag for Power-On reset 0 No power-on event has occurred since the last clear (due to either a software clear or a low-voltage detection) 1 A power-on event has occurred
F_LVD27_VREG	Flag for 2.7 V low-voltage detected (VREG) 0 No 2.7 V low-voltage detected (VREG) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7 V low-voltage detected (VREG) event has occurred
F_LVD27	Flag for 2.7 V low-voltage detected 0 No 2.7 V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7 V low-voltage detected event has occurred
F_SWT	Flag for software watchdog timer 0 No software watchdog timer event has occurred since either the last clear or the last power-on reset assertion 1 A software watchdog timer event has occurred
F_LVD12_PD1	Flag for 1.2 V low-voltage detected (power domain #1) 0 No 1.2 V low-voltage detected (power domain #1) event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2 V low-voltage detected (power domain #1) event has occurred
F_LVD12_PD0	Flag for 1.2 V low-voltage detected (power domain #0) 0 No 1.2 V low-voltage detected (power domain #0) event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2 V low-voltage detected (power domain #0) event has occurred

*Note:* The F\_POR flag is automatically cleared on a 1.2 V low-voltage detected (power domain #0 or #1) or a 2.7 V low-voltage detected. This means that if the power-up sequence is not monotonic (i.e the voltage rises and then drops enough to trigger a low-voltage detection), the F\_POR flag may not be set but instead the <register>F\_LVD12\_PD0, <register>F\_LVD12\_PD1, or <register>F\_LVD27 flag is set on exiting the reset sequence. Therefore, if the F\_POR, <register>F\_LVD12\_PD0, <register>F\_LVD12\_PD1, or <register>F\_LVD27 flags are set on reset exit, software should interpret the reset cause as power-on.

*Note:* In contrast to all other reset sources, the 1.2 V low-voltage detected (power domain #0) event is captured on its deassertion. Therefore, the status bit F\_LVD12\_PD0 is also asserted on the reset's deassertion. In case an alternate event is selected, the SAFE mode or interrupt request are similarly asserted on the reset's deassertion.

### 9.3.1.3 Functional Event Reset Disable Register (RGM\_FERD)

This register provides dedicated bits to disable functional reset sources. When a functional reset source is disabled, the associated functional event will trigger either a SAFE mode request or an interrupt request (see [Section 9.3.1.5: Functional Event Alternate Request Register \(RGM\\_FEAR\)](#)). It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode. Each byte can be written only once after power-on reset.

Address 0xC3FE\_4004

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D_EXR	0	0	0	0	0	0	D_FLASH	D_LVD45	D_CMU_FHL	D_CMU_OLR	D_FMPLL	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 77. Functional Event Reset Disable Register (RGM\_FERD)

Table 82. Functional Event Reset Disable Register (RGM\_FERD) field descriptions

Field	Description
D_EXR	Disable External Reset 0 An external reset event triggers a reset sequence 1 An external reset event generates a SAFE mode request
D_FLASH	Disable code or data flash fatal error 0 A code or data flash fatal error event triggers a reset sequence 1 A code or data flash fatal error event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_FLASH
D_LVD45	Disable 4.5 V low-voltage detected 0 A 4.5 V low-voltage detected event triggers a reset sequence 1 A 4.5 V low-voltage detected event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_LVD45
D_CMU_FHL	Disable CMU clock frequency higher/lower than reference 0 A CMU clock frequency higher/lower than reference event triggers a reset sequence 1 A CMU clock frequency higher/lower than reference event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CMU_FHL
D_CMU_OLR	Disable FXOSC frequency lower than reference 0 A FXOSC frequency lower than reference event triggers a reset sequence 1 A FXOSC frequency lower than reference event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CMU_OLR
D_FMPLL	Disable FMPLL fail 0 A FMPLL fail event triggers a reset sequence 1 A FMPLL fail event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_FMPLL
D_CHKSTOP	Disable checkstop reset 0 A checkstop reset event triggers a reset sequence 1 A checkstop reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CHKSTOP
D_SOFT	Disable software reset 0 A software reset event triggers a reset sequence 1 A software reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_SOFT

**Table 82. Functional Event Reset Disable Register (RGM\_FERD) field descriptions (continued)**

Field	Description
D_CORE	Disable debug control core reset 0 A debug control core reset event triggers a reset sequence 1 A debug control core reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CORE
D_JTAG	Disable JTAG initiated reset 0 A JTAG initiated reset event triggers a reset sequence 1 A JTAG initiated reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_JTAG

**9.3.1.4 Destructive Event Reset Disable Register (RGM\_DERD)**

Address 0xC3FE\_4006

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	D_LVD27_VREG	D_LVD27	D_SWT	D_LVD12_PD1	D_LVD12_PD0
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 78. Destructive Event Reset Disable Register (RGM\_DERD)**

This register provides dedicated bits to disable particular destructive reset sources. When a destructive reset source is disabled, the associated destructive event will trigger either a safe mode request or an interrupt request (see [Section 9.3.1.6: Destructive Event Alternate Request Register \(RGM\\_DEAR\)](#)).

**Table 83. Destructive Event Reset Disable Register (RGM\_DERD) field descriptions**

Field	Description
D_LVD27_VREG	Disable 2.7 V low-voltage detected (VREG) 0 A 2.7 V low-voltage detected (VREG) event triggers a reset sequence 1 A 2.7 V low-voltage detected (VREG) event generates either a SAFE mode or an interrupt request depending on the value of RGM_DEAR.AR_LVD27_VREG
D_LVD27	Disable 2.7 V low-voltage detected 0 A 2.7 V low-voltage detected event triggers a reset sequence 1 A 2.7 V low-voltage detected event generates either a SAFE mode or an interrupt request depending on the value of RGM_DEAR.AR_LVD27
D_SWT	Disable software watchdog timer 0 A software watchdog timer event triggers a reset sequence 1 A software watchdog timer event generates either a SAFE mode or an interrupt request depending on the value of RGM_DEAR.ep

**Table 83. Destructive Event Reset Disable Register (RGM\_DERD) field descriptions (continued)**

Field	Description
D_LVD12_PD1	Disable 1.2 V low-voltage detected (power domain #1) 0 A 1.2 V low-voltage detected (power domain #1) event triggers a reset sequence 1 A 1.2 V low-voltage detected (power domain #1) event generates either a SAFE mode or an interrupt request depending on the value of RGM_DEAR.AR_LVD12_PD1
D_LVD12_PD0	Disable 1.2 V low-voltage detected (power domain #0) 0 A 1.2 V low-voltage detected (power domain #0) event triggers a reset sequence 1 A 1.2 V low-voltage detected (power domain #0) event generates either a SAFE mode or an interrupt request depending on the value of RGM_DEAR.AR_LVD12_PD0

**9.3.1.5 Functional Event Alternate Request Register (RGM\_FEAR)**

Address 0xC3FE\_4010

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AR_EXR	0	0	0	0	0	0	AR_FLASH	R_LVD45	AR_CMU_FHL	AR_CMU_OLR	AR_FMPLL	AR_CHKSTOP	AR_SOFT	AR_CORE	AR_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 79. Functional Event Alternate Request Register (RGM\_FEAR)**

This register defines an alternate request to be generated when a reset on a functional event has been disabled. The alternate request can be either a SAFE mode request to MC\_ME or an interrupt request to the system. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode.

**Table 84. Functional Event Alternate Request Register (RGM\_FEAR) field descriptions**

Field	Description
AR_EXR	Alternate Request for External Reset 0 Generate a SAFE mode request on an <b>external reset</b> event if the reset is disabled 1 Generate an interrupt request on an <b>external reset</b> event if the reset is disabled
AR_FLASH	Alternate Request for code or data flash fatal error 0 Generate a SAFE mode request on a code or data flash fatal error event if the reset is disabled 1 Generate an interrupt request on a code or data flash fatal error event if the reset is disabled
AR_LVD45	Alternate Request for 4.5 V low-voltage detected 0 Generate a SAFE mode request on a 4.5 V low-voltage detected event if the reset is disabled 1 Generate an interrupt request on a 4.5 V low-voltage detected event if the reset is disabled

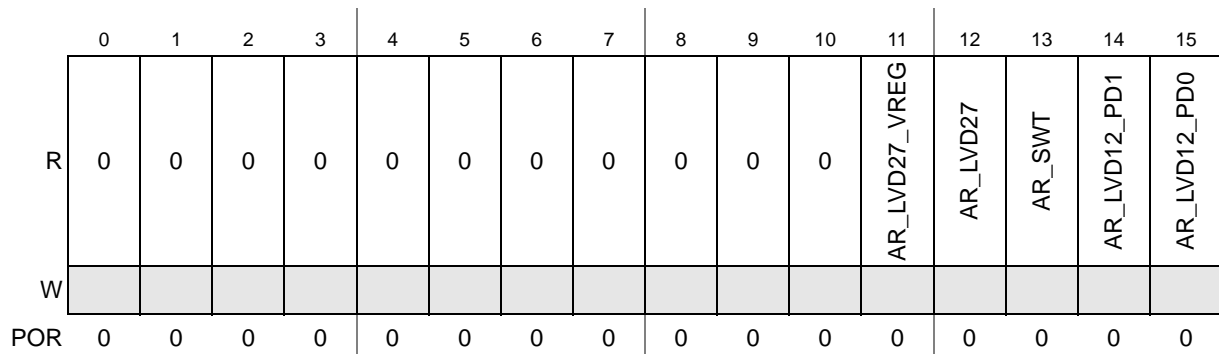
**Table 84. Functional Event Alternate Request Register (RGM\_FEAR) field descriptions**

Field	Description
AR_CMU_FHL	Alternate Request for CMU clock frequency higher/lower than reference 0 Generate a SAFE mode request on a CMU clock frequency higher/lower than reference event if the reset is disabled 1 Generate an interrupt request on a CMU clock frequency higher/lower than reference event if the reset is disabled
AR_CMU_OLR	Alternate Request for FXOSC frequency lower than reference 0 Generate a SAFE mode request on a FXOSC frequency lower than reference event if the reset is disabled 1 Generate an interrupt request on a FXOSC frequency lower than reference event if the reset is disabled For the case when RGM_FERD[D_CMU_OLR] = 1 & RGM_FEAR[AR_CMU_OLR] = 1, an RGM interrupt will not be generated for an FXOSC failure when the system clock = FXOSC as there will be no system clock to execute the interrupt service routine. However, the interrupt service routine will be executed if the FXOSC recovers at some point. The recommended use case for this feature is when the system clock = FIRC or FMPLL.
AR_FMPLL	Alternate Request for FMPLL fail 0 Generate a SAFE mode request on a FMPLL fail event if the reset is disabled 1 Generate an interrupt request on a FMPLL fail event if the reset is disabled
AR_CHKSTOP	Alternate Request for checkstop reset 0 Generate a SAFE mode request on a checkstop reset event if the reset is disabled 1 Generate an interrupt request on a checkstop reset event if the reset is disabled
AR_SOFT	Alternate Request for software reset 0 Generate a SAFE mode request on a software reset event if the reset is disabled 1 Generate an interrupt request on a software reset event if the reset is disabled
AR_CORE	Alternate Request for debug control core reset 0 Generate a SAFE mode request on a debug control core reset event if the reset is disabled 1 Generate an interrupt request on a debug control core reset event if the reset is disabled
AR_JTAG	Alternate Request for JTAG initiated reset 0 Generate a SAFE mode request on a JTAG initiated reset event if the reset is disabled 1 Generate an interrupt request on a JTAG initiated reset event if the reset is disabled

**9.3.1.6 Destructive Event Alternate Request Register (RGM\_DEAR)**

Address: 0xC3FE\_4012

Access: Read



**Figure 80. Destructive Event Alternate Request Register (RGM\_DEAR)**





This register defines an alternate request to be generated when a reset on a destructive event has been disabled. The alternate request can be either a SAFE mode request to MC\_ME or an interrupt request to the system.

**Table 85. Destructive Event Alternate Request Register (RGM\_DEAR) field descriptions**

Field	Description
AR_LVD27_VREG	Alternate Request for 2.7 V low-voltage detected (VREG) 0 Generate a SAFE mode request on a 2.7 V low-voltage detected (VREG) event if the reset is disabled 1 Generate an interrupt request on a 2.7 V low-voltage detected (VREG) event if the reset is disabled
AR_LVD27	Alternate Request for 2.7 V low-voltage detected 0 Generate a SAFE mode request on a 2.7 V low-voltage detected event if the reset is disabled 1 Generate an interrupt request on a 2.7 V low-voltage detected event if the reset is disabled
AR_SWT	Alternate Request for software watchdog timer 0 Generate a SAFE mode request on a software watchdog timer event if the reset is disabled 1 Generate an interrupt request on a software watchdog timer event if the reset is disabled
AR_LVD12_PD1	Alternate Request for 1.2 V low-voltage detected (power domain #1) 0 Generate a SAFE mode request on a 1.2 V low-voltage detected (power domain #1) event if the reset is disabled 1 Generate an interrupt request on a 1.2 V low-voltage detected (power domain #1) event if the reset is disabled
AR_LVD12_PD0	Alternate Request for 1.2 V low-voltage detected (power domain #0) 0 Generate a SAFE mode request on a 1.2 V low-voltage detected (power domain #0) event if the reset is disabled 1 Generate an interrupt request on a 1.2 V low-voltage detected (power domain #0) event if the reset is disabled

**9.3.1.7 Functional Event Short Sequence Register (RGM\_FESS)**

Address 0xC3FE\_4018

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS_EXR	0	0	0	0	0	0	SS_FLASH	SS_LVD45	SS_CMU_FHL	SS_CMU_OLR	SS_FMPLL	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 81. Functional Event Short Sequence Register (RGM\_FESS)**

This register defines which reset sequence will be done when a functional reset sequence is triggered. The functional reset sequence can either start from PHASE1 or from PHASE3, skipping PHASE1 and PHASE2.

*Note:* This could be useful for fast reset sequence, for example to skip flash reset.



It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

**Table 86. Functional Event Short Sequence Register (RGM\_FESS) field descriptions**

Field	Description
SS_EXR	Short Sequence for External Reset 0 The reset sequence triggered by an <b>external</b> reset event will start from PHASE1 1 The reset sequence triggered by an <b>external</b> reset event will start from PHASE3, skipping PHASE1 and PHASE2
SS_FLASH	Short Sequence for code or data flash fatal error 0 The reset sequence triggered by a code or data flash fatal error event will start from PHASE1 1 The reset sequence triggered by a code or data flash fatal error event will start from PHASE3, skipping PHASE1 and PHASE2
SS_LVD45	Short Sequence for 4.5 V low-voltage detected 0 The reset sequence triggered by a 4.5 V low-voltage detected event will start from PHASE1 1 The reset sequence triggered by a 4.5 V low-voltage detected event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CMU_FHL	Short Sequence for CMU clock frequency higher/lower than reference 0 The reset sequence triggered by a CMU clock frequency higher/lower than reference event will start from PHASE1 1 The reset sequence triggered by a CMU clock frequency higher/lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CMU_OLR	Short Sequence for FXOSC frequency lower than reference 0 The reset sequence triggered by a FXOSC frequency lower than reference event will start from PHASE1 1 The reset sequence triggered by a FXOSC frequency lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2
SS_FMPLL	Short Sequence for FMPLL fail 0 The reset sequence triggered by a FMPLL fail event will start from PHASE1 1 The reset sequence triggered by a FMPLL fail event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CHKSTOP	Short Sequence for checkstop reset 0 The reset sequence triggered by a checkstop reset event will start from PHASE1 1 The reset sequence triggered by a checkstop reset event will start from PHASE3, skipping PHASE1 and PHASE2
SS_SOFT	Short Sequence for software reset 0 The reset sequence triggered by a software reset event will start from PHASE1 1 The reset sequence triggered by a software reset event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CORE	Short Sequence for debug control core reset 0 The reset sequence triggered by a debug control core reset event will start from PHASE1 1 The reset sequence triggered by a debug control core reset event will start from PHASE3, skipping PHASE1 and PHASE2
SS_JTAG	Short Sequence for JTAG initiated reset 0 The reset sequence triggered by a JTAG initiated reset event will start from PHASE1 1 The reset sequence triggered by a JTAG initiated reset event will start from PHASE3, skipping PHASE1 and PHASE2

9.3.1.8 STANDBY Reset Sequence Register (RGM\_STDBY)

Address 0xC3FE\_401A Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	0	0	0	0	0	0	0	0	BOOT_FROM_BKP_RAM	0	0	0	0	0	0	0	0
W																	
reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 82. STANDBY Reset Sequence Register (RGM\_STDBY)

This register defines the reset sequence to be applied on STANDBY mode exit. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode.

Table 87. STANDBY Reset Sequence Register (RGM\_STDBY) field descriptions

Field	Description
BOOT_FROM_BKP_RAM	<b>Boot from Backup RAM indicator</b> — This bit indicates whether the system will boot from backup RAM or flash out of STANDBY exit. 0 Boot from default boot location on STANDBY exit 1 Boot from backup RAM on STANDBY exit

Note: This register is reset on any enabled 'destructive' or 'functional' reset event.

9.3.1.9 Functional Bidirectional Reset Enable Register (RGM\_FBRE)

Address 0xC3FE\_401C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BE_EXR	0	0	0	0	0	0	BE_FLASH	BE_LVD45	BE_CMU_FHL	BE_CMU_OLR	BE_FMPPLL	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 83. Functional Bidirectional Reset Enable Register (RGM\_FBRE)

This register enables the generation of an external reset on functional reset. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 88. Functional Bidirectional Reset Enable Register (RGM\_FBRE) field descriptions

Field	Description
BE_EXR	Bidirectional Reset Enable for External Reset 0 RESET is asserted on an <b>external</b> reset event if the reset is enabled 1 RESET is not asserted on an <b>external</b> reset event
BE_FLASH	Bidirectional Reset Enable for code or data flash fatal error 0 RESET is asserted on a code or data flash fatal error event if the reset is enabled 1 RESET is not asserted on a code or data flash fatal error event
BE_LVD45	Bidirectional Reset Enable for 4.5 V low-voltage detected 0 RESET is asserted on a 4.5 V low-voltage detected event if the reset is enabled 1 RESET is not asserted on a 4.5 V low-voltage detected event
BE_CMU_FHL	Bidirectional Reset Enable for CMU clock frequency higher/lower than reference 0 RESET is asserted on a CMU clock frequency higher/lower than reference event if the reset is enabled 1 RESET is not asserted on a CMU clock frequency higher/lower than reference event
BE_CMU_OLR	Bidirectional Reset Enable for FXOSC frequency lower than reference 0 RESET is asserted on a FXOSC frequency lower than reference event if the reset is enabled 1 RESET is not asserted on a FXOSC frequency lower than reference event
BE_FMPLL	Bidirectional Reset Enable for FMPLL fail 0 RESET is asserted on a FMPLL fail event if the reset is enabled 1 RESET is not asserted on a FMPLL fail event
BE_CHKSTOP	Bidirectional Reset Enable for checkstop reset 0 RESET is asserted on a checkstop reset event if the reset is enabled 1 RESET is not asserted on a checkstop reset event
BE_SOFT	Bidirectional Reset Enable for software reset 0 RESET is asserted on a software reset event if the reset is enabled 1 RESET is not asserted on a software reset event
BE_CORE	Bidirectional Reset Enable for debug control core reset 0 RESET is asserted on a debug control core reset event if the reset is enabled 1 RESET is not asserted on a debug control core reset event
BE_JTAG	Bidirectional Reset Enable for JTAG initiated reset 0 RESET is asserted on a JTAG initiated reset event if the reset is enabled 1 is not asserted on a JTAG initiated reset event

## 9.4 Functional description

### 9.4.1 Reset State Machine

The main role of MC\_RGM is the generation of the reset sequence which ensures that the correct parts of the device are reset based on the reset source event. This is summarized in [Table 89](#).

Table 89. MC\_RGM Reset Implications

Source	What Gets Reset	External Reset Assertion	Boot Mode Capture
Power-on Reset	All	Yes	Yes
'destructive' Resets	All except some clock/reset management	Yes	Yes
External Reset	All except some clock/reset management and debug	Yes	Yes
'functional' Resets	All except some clock/reset management and debug	Programmable <sup>(1)</sup>	Programmable <sup>(2)</sup>
Shortened 'functional' Resets <sup>(3)</sup>	Flip-flops except some clock/reset management	Programmable <sup>(1)</sup>	Programmable <sup>(2)</sup>

1. the assertion of the external reset is controlled via the RGM\_FBRE register.
2. the boot mode is captured if the external reset is asserted.
3. the short sequence is enabled via the RGM\_FESS register.

**Note:** *JTAG logic has its own independent reset control and is not controlled by the MC\_RGM in any way.*

The reset sequence is comprised of five phases managed by a state machine, which ensures that all phases are correctly processed through waiting for a minimum duration and until all processes that need to occur during that phase have been completed before proceeding to the next phase.

The state machine used to produce the reset sequence is shown in [Figure 84](#).

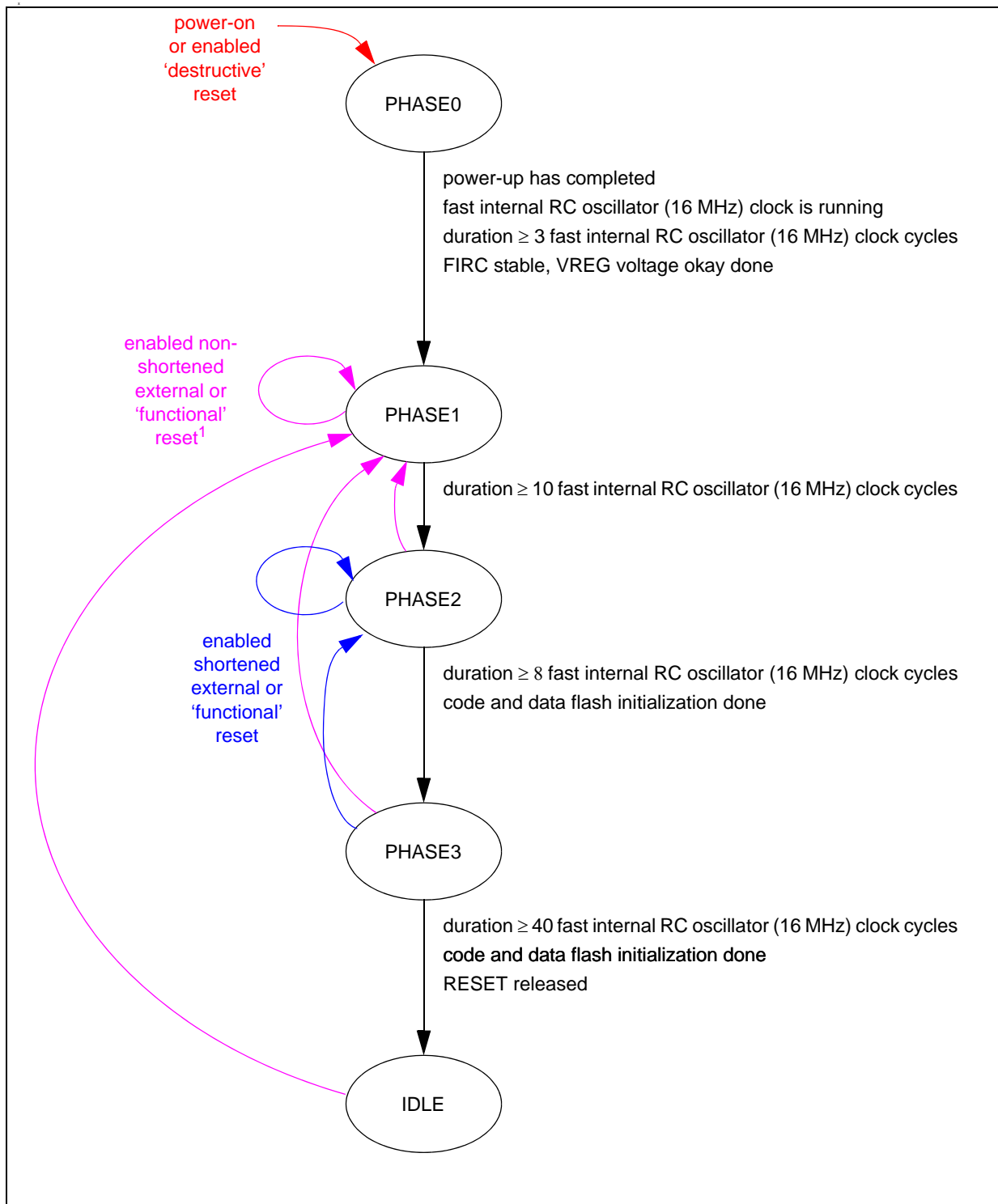
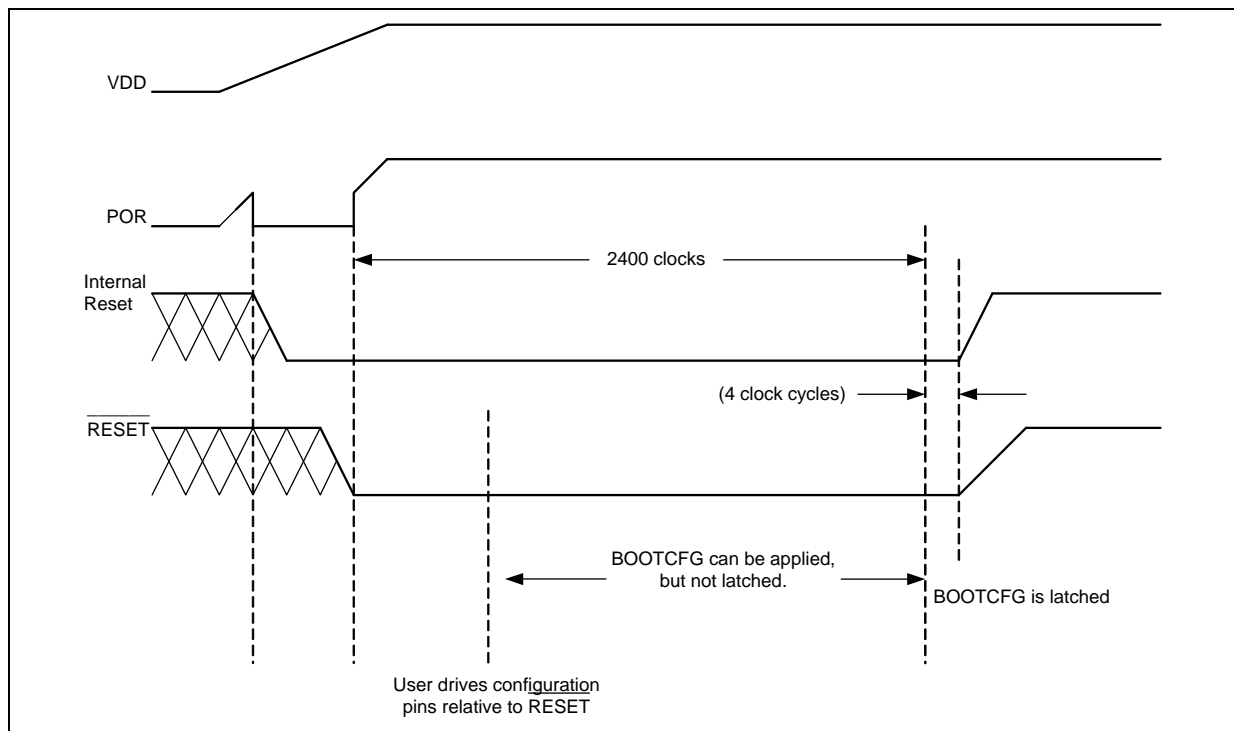


Figure 84. MC\_RGM State Machine

Figure 85 describes how the device behaves during the startup.



**Figure 85. Reset configuration timing**

#### 9.4.1.1 PHASE0 phase

This phase is entered immediately from any phase on a power-on or enabled 'destructive' reset event. The reset state machine exits PHASE0 and enters PHASE1 on verification of the following:

- power-up has completed
- fast internal RC oscillator (16 MHz) clock is running
- all enabled 'destructive' resets have been processed
- all processes that need to be done in PHASE0 are completed
  - FIRC stable, VREG voltage okay
- a minimum of 3 fast internal RC oscillator (16 MHz) clock cycles have elapsed since power-up completion and the last enabled 'destructive' reset event

#### 9.4.1.2 PHASE1 phase

This phase is entered either on exit from PHASE0 or immediately from PHASE2, PHASE3, or IDLE on a non-masked external or 'functional' reset event if it has not been configured to trigger a 'short' sequence. The reset state machine exits PHASE1 and enters PHASE2 on verification of the following:

- all enabled, non-shortened 'functional' resets have been processed
- a minimum of 10 fast internal RC oscillator (16 MHz) clock cycles have elapsed since the last enabled external or non-shortened 'functional' reset event

### 9.4.1.3 PHASE2 phase

This phase is entered on exit from PHASE1. The reset state machine exits PHASE2 and enters PHASE3 on verification of the following:

- all processes that need to be done in PHASE2 are completed
  - code and data flash initialization
- a minimum of 8 fast internal RC oscillator (16 MHz) clock cycles have elapsed since entering PHASE2

### 9.4.1.4 PHASE3 phase

This phase is entered either on exit from PHASE2 or immediately from IDLE on an enabled, shortened 'functional' reset event. The reset state machine exits PHASE3 and enters IDLE on verification of the following:

- all processes that need to be done in PHASE3 are completed
  - code and data flash initialization
- a minimum of 40 fast internal RC oscillator (16 MHz) clock cycles have elapsed since the last enabled, shortened 'functional' reset event

### 9.4.1.5 IDLE phase

This is the final phase and is entered on exit from PHASE3. When this phase is reached, the MC\_RGM releases control of the system to the platform and waits for new reset events that can trigger a reset sequence.

## 9.4.2 Destructive resets

A 'destructive' reset indicates that an event has occurred after which critical register or memory content can no longer be guaranteed.

The status flag associated with a given 'destructive' reset event (RGM\_DES.F\_<destructive reset> bit) is set when the 'destructive' reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously, and it is software's responsibility to determine which reset source is the most critical for the application.

The 'destructive' reset can be optionally disabled by writing bit RGM\_DERD.D\_<destructive reset>.

*Note:* The RGM\_DERD register can be written only once between two power-on reset events.

The device's low-voltage detector threshold ensures that, when 1.2 V low-voltage detected (power domain #0) is enabled, the supply is sufficient to have the destructive event correctly propagated through the digital logic. Therefore, if a given 'destructive' reset is enabled, the MC\_RGM ensures that the associated reset event will be correctly triggered to the full system. However, if the given 'destructive' reset is disabled and the voltage goes below the digital functional threshold, functionality can no longer be ensured, and the reset may or may not be asserted.

An enabled destructive reset will trigger a reset sequence starting from the beginning of PHASE0.



### 9.4.3 External reset

The MC\_RGM manages the external reset coming from RESET. The detection of a falling edge on RESET will start the reset sequence from the beginning of PHASE1.

The status flag associated with the external reset falling edge event (RGM\_FES.F\_EXR bit) is set when the external reset is asserted and the power-on reset is not asserted.

The external reset can optionally be disabled by writing bit RGM\_FERD.D\_EXR.

*Note:* The RGM\_FERD register can be written only once between two power-on reset events.

An enabled external reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM\_FESS register enables the further configuring of the reset sequence triggered by the external reset. When RGM\_FESS.SS\_EXR is set, the external reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially when an external reset should not reset the flash.

The MC\_RGM may also assert the external reset if the reset sequence was triggered by one of the following:

- a power-on reset
- a 'destructive' reset event
- an external reset event
- a 'functional' reset event configured via the RGM\_FBRE register to assert the external reset

In this case, the external reset is asserted until the end of PHASE3.

### 9.4.4 Functional resets

A 'functional' reset indicates that an event has occurred after which it can be guaranteed that critical register and memory content is still intact.

The status flag associated with a given 'functional' reset event (RGM\_FES.F\_<functional reset> bit) is set when the 'functional' reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously, and it is software's responsibility to determine which reset source is the most critical for the application.

The 'functional' reset can be optionally disabled by software writing bit RGM\_FERD.D\_<functional reset>.

*Note:* The RGM\_FERD register can be written only once between two power-on reset events.

An enabled functional reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM\_FESS register enables the further configuring of the reset sequence triggered by a functional reset. When RGM\_FESS.SS\_<functional reset> is set, the associated 'functional' reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially in case a functional reset should not reset the flash module.

### 9.4.5 STANDBY entry sequence

STANDBY mode can be entered only when the MC\_RGM is in IDLE. On STANDBY entry, the MC\_RGM moves to PHASE1. The minimum duration counter in PHASE1 does not start

until STANDBY mode is exited. On entry to PHASE1 due to STANDBY mode entry, the resets for all power domains except power domain #0 are asserted. During this time, RESET is not asserted as the external reset can act as a wakeup for the device.

There is an option to keep the flash inaccessible and in low-power mode on STANDBY exit by configuring the DRUN mode before STANDBY entry so that the flash is in power-down or low-power mode. If the flash is to be inaccessible, the PHASE2 and PHASE3 states do not wait for the flash to complete initialization before exiting, and the reset to the flash remains asserted.

See the MC\_ME chapter for details on the STANDBY and DRUN modes.

#### 9.4.6 Alternate event generation

The MC\_RGM provides alternative events to be generated on reset source assertion. When a reset source is asserted, the MC\_RGM normally enters the reset sequence. Alternatively, it is possible for each reset source event (except the power-on reset event) to be converted from a reset to either a SAFE mode request issued to the MC\_ME or to an interrupt request issued to the core.

Alternate event selection for a given reset source is made via the RGM\_F/DERD and RGM\_F/DEAR registers as shown in [Table 90](#).

**Table 90. MC\_RGM alternate event selection**

RGM_F/DERD bit value	RGM_F/DEAR bit value	Generated event
0	X	Reset
1	0	SAFE mode request
1	1	Interrupt request

The alternate event is cleared by deasserting the source of the request (i.e. at the reset source that caused the alternate request) and also clearing the appropriate RGM\_F/DES status bit.

*Note:* Alternate requests (SAFE mode as well as interrupt requests) are generated asynchronously.

*Note:* If a masked 'destructive' reset event which is configured to generate a SAFE mode/interrupt request occurs during PHASE0, it is ignored, and the MC\_RGM will not send any safe mode/interrupt request to the MC\_ME. The same is true for masked 'functional' reset events during PHASE1.

#### 9.4.7 Boot mode capturing

The MC\_RGM samples PA[9:8] whenever RESET is asserted until five FIRC (16 MHz internal RC oscillator) clock cycles before its deassertion edge. The result of the sampling is used at the beginning of reset PHASE3 for boot mode selection and is retained after RESET has been deasserted for subsequent boots after reset sequences during which RESET is not asserted.

*Note:* In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value the entire time that RESET is asserted.

RESET can be asserted as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins (see [Table 89](#) for details).

## 10 Power Control Unit (MC\_PCU)

### 10.1 Introduction

#### 10.1.1 Overview

The power control unit (MC\_PCU) is used to reduce the overall SoC power consumption. Power can be saved by disconnecting parts of the SoC from the power supply via a power switching device. The SoC is grouped into multiple parts having this capability which are called "power domains".

When a power domain is disconnected from the supply, the power consumption is reduced to zero in that domain. Any status information of such a power domain is lost. When re-connecting a power domain to the supply voltage, the domain draws an increased current until the power domain reaches its operational voltage.

Power domains are controlled on a device mode basis. For each mode, software can configure whether a power domain is connected to the supply voltage (power-up state) or disconnected (power-down state). Maximum power saving is reached by entering the STANDBY mode.

On each mode change request, the MC\_PCU evaluates the power domain settings in the power domain configuration registers and initiates a power-down or a power-up sequence for each individual power domain. The power-up/down sequences are handled by finite state machines to ensure a smooth and safe transition from one power state to the other.

Exiting the STANDBY mode can only be done via a system wakeup event as all power domains other than power domain #0 are in the power-down state.

In addition, the MC\_PCU acts as a bridge for mapping the VREG peripheral to the MC\_PCU address space.

*Figure 86* depicts the MC\_PCU block diagram.

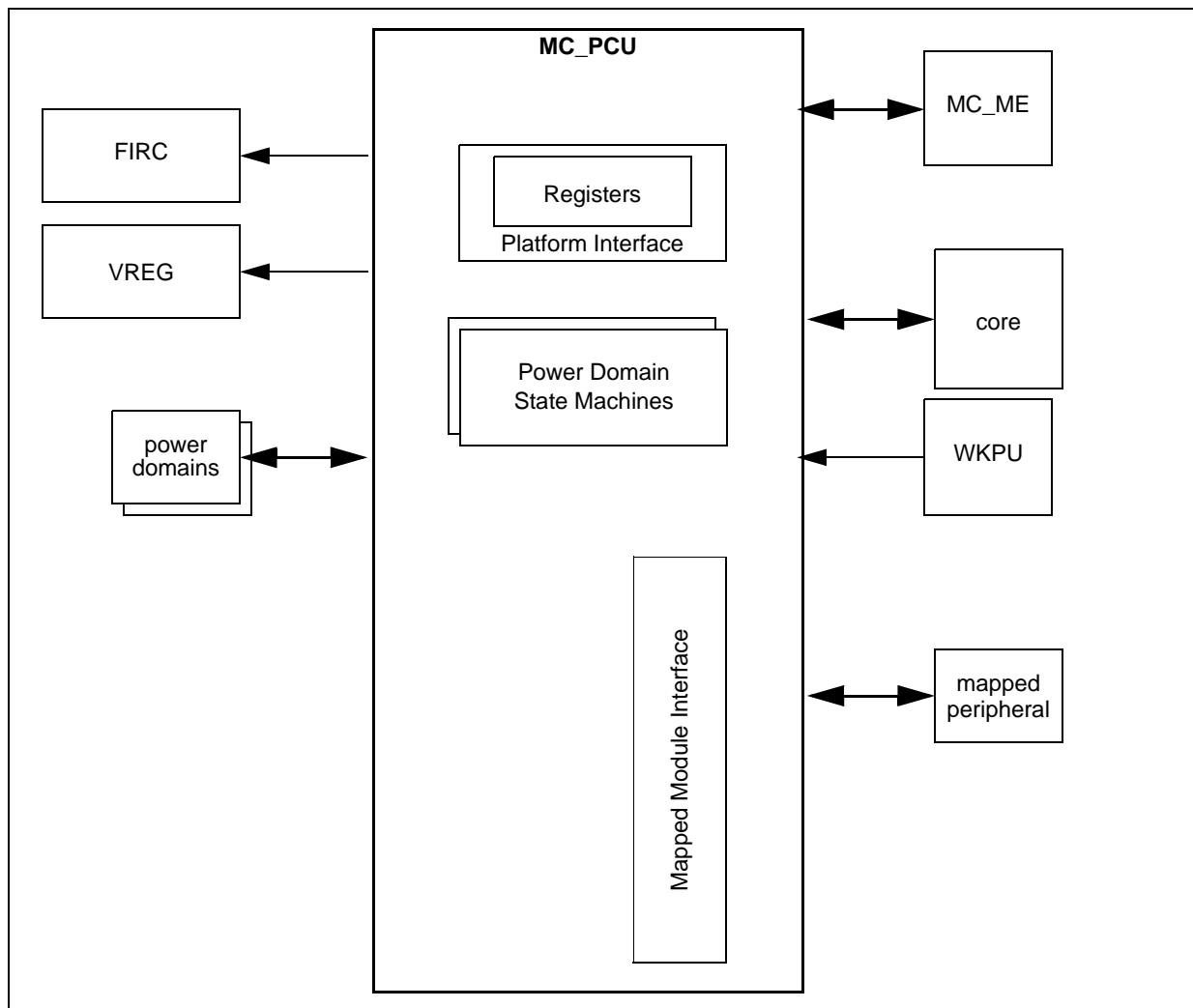


Figure 86. MC\_PCU Block Diagram

### 10.1.2 Features

The MC\_PCU includes the following features:

- support for 4 power domains
- support for device modes RESET, DRUN, SAFE, TEST, RUN0...3, HALT, STOP, and STANDBY (for further mode details, please see the MC\_ME chapter)
- power states updating on each mode change and on system wakeup
- a handshake mechanism for power state changes thus guaranteeing operable voltage
- maps the VREG registers to the MC\_PCU address space

### 10.1.3 Modes of operation

The MC\_PCU is available in all device modes.

## 10.2 External signal description

The MC\_PCU has no connections to any external pins.

## 10.3 Memory map and register definition

Table 91. MC\_PCU register description

Address	Name	Description	Size	Access			Location
				Normal	Supervisor	Test	
0xC3FE_8000	PCU_PCONF0	Power Domain #0 Configuration	word	read	read	read	<a href="#">on page 256</a>
0xC3FE_8004	PCU_PCONF1	Power Domain #1 Configuration	word	read	read	read	<a href="#">on page 257</a>
0xC3FE_8008	PCU_PCONF2	Power Domain #2 Configuration	word	read	read/write	read/write	<a href="#">on page 258</a>
0xC3FE_800C	PCU_PCONF3 <sup>(1)</sup>	Power Domain #3 Configuration	word	read	read/write	read/write	<a href="#">on page 258</a>
0xC3FE_8040	PCU_PSTAT	Power Domain Status Register	word	read	read	read	<a href="#">on page 259</a>

1. Though this register is implemented, programming this register will have no impact on device operation.

*Note:* Any access to unused registers as well as write accesses to read-only registers will: not change register content cause a transfer error

Table 92. MC\_PCU memory map

Address	Name	Bit																
		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
0xC3FE_8000	PCU_PCONF0	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																
		R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_8004	PCU_PCONF1	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																
		R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																

Table 92. MC\_PCU memory map (continued)

Address	Name	Bit positions																
		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15	
0xC3FE_8008	PCU_PCONF2	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																
		R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_800C	PCU_PCONF3	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																
		R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_8010 ... 0xC3FE_803C	reserved																	
0xC3FE_8040	PCU_PSTAT	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	PD3	PD2	PD1	PD0
		W																
0x044 ... 0x07C	reserved																	
0xC3FE_8080 ... 0xC3FE_80FC	VREG registers																	
0xC3FE_8100 ... 0xC3FE_BFFC	reserved																	

### 10.3.1 Register descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the PD0 field of the PCU\_PSTAT register may be accessed as a word at address 0xC3FE\_8040, as a half-word at address 0xC3FE\_8042, or as a byte at address 0xC3FE\_8043.

**10.3.1.1 Power Domain #0 Configuration Register (PCU\_PCONF0)**

Address 0xC3FE\_8000

Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1

**Figure 87. Power Domain #0 Configuration Register (PCU\_PCONF0)**

This register defines for power domain #0 whether it is on or off in each device mode. As power domain #0 is the always-on power domain (and includes the MC\_PCU), none of its bits are programmable. This register is available for completeness reasons.

**Table 93. Power Domain Configuration Register field descriptions**

Field	Description
RST	Power domain control during RESET mode 0 Power domain off 1 Power domain on
TEST	Power domain control during TEST mode 0 Power domain off 1 Power domain on
SAFE	Power domain control during SAFE mode 0 Power domain off 1 Power domain on
DRUN	Power domain control during DRUN mode 0 Power domain off 1 Power domain on
RUN0	Power domain control during RUN0 mode 0 Power domain off 1 Power domain on
RUN1	Power domain control during RUN1 mode 0 Power domain off 1 Power domain on
RUN2	Power domain control during RUN2 mode 0 Power domain off 1 Power domain on



**Table 93. Power Domain Configuration Register field descriptions (continued)**

Field	Description
RUN3	Power domain control during RUN3 mode 0 Power domain off 1 Power domain on
HALT	Power domain control during HALT mode 0 Power domain off 1 Power domain on
STOP	Power domain control during STOP mode 0 Power domain off 1 Power domain on
STBY0	Power domain control during STANDBY mode 0 Power domain off 1 Power domain on

**10.3.1.2 Power Domain #1 Configuration Register (PCU\_PCONF1)**

Address 0xC3FE\_8004

Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1

**Figure 88. Power Domain #1 Configuration Register (PCU\_PCONF1)**

This register defines for power domain #1 whether it is on or off in each device mode. The bit field description is the same as in [Table 93](#). As the platform, clock generation, and mode control reside in power domain #1, this power domain is only powered down during the STANDBY mode. Therefore, none of the bits is programmable. This register is available for completeness reasons.

The difference between PCU\_PCONF0 and PCU\_PCONF1 is the reset value of the STBY0 bit: During the STANDBY mode, power domain #1 is disconnected from the power supply, and therefore PCU\_PCONF1.STBY0 is always '0'. Power domain #0 is always on, and therefore PCU\_PCONF0.STBY0 is '1'.

For further details about STANDBY mode, please see [Section 10.4.4.2: STANDBY mode transition](#).

**10.3.1.3 Power Domain #2...3 Configuration Registers (PCU\_PCONF2...3)**

Address 0xC3FE\_8008 Access: User read/write, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1

**Figure 89. Power Domain #2 Configuration Register (PCU\_PCONF2)**

Address 0xC3FE\_800C Access: User read/write, Supervisor read/write, Test read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1

**Figure 90. Power Domain #3 Configuration Register (PCU\_PCONF3)**

These registers define for each power domain #2 through #3 whether it is on or off in each device mode. The bit field description is the same as in [Table 93](#).

**10.3.1.4 Power Domain Status Register (PCU\_PSTAT)**

Address 0xC3FE\_8040 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PD3	PD2	PD1	PD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

**Figure 91. Power Domain Status Register (PCU\_PSTAT)**

This register reflects the power status of all available power domains.

**Table 94. Power Domain Status Register (PCU\_PSTAT) field descriptions**

Field	Description
PD $n$	Power status for power domain # $n$ 0 Power domain is inoperable 1 Power domain is operable

**10.4 Functional description**

**10.4.1 General**

The MC\_PCU controls all available power domains on a device mode basis. The PCU\_PCONFn registers specify during which system/user modes a power domain is powered up. The power state for each individual power domain is reflected by the bits in the PCU\_PSTAT register.

On a mode change, the MC\_PCU evaluates which power domain(s) must change power state. The power state is controlled by a state machine (FSM) for each individual power domain which ensures a clean and safe state transition.

**10.4.2 Reset / Power-On Reset**

After any reset, the SoC will transition to the RESET mode during which all power domains are powered up (see the MC\_ME chapter). Once the reset sequence has been completed, the DRUN mode is entered and software can begin the MC\_PCU configuration.

**10.4.3 MC\_PCU configuration**

Per default, all power domains are powered in all modes other than STANDBY. Software can change the configuration for each power domain on a mode basis by programming the PCU\_PCONFn registers.

Each power domain which is powered down is held in a reset state. Read/write accesses to peripherals in those power domains will result in a transfer error.

### 10.4.4 Mode transitions

On a mode change requested by the MC\_ME, the MC\_PCU evaluates the power configurations for all power domains. It compares the settings in the PCU\_PCONF<sub>n</sub> registers for the new mode with the settings for the current mode. If the configuration for a power domain differs between the modes, a power state change request is generated. These requests are handled by a finite state machine to ensure a smooth and safe transition from one power state to another.

#### 10.4.4.1 DRUN, SAFE, TEST, RUN0...3, HALT, and STOP mode transition

The DRUN, SAFE, TEST, RUN0...3, HALT, and STOP modes allow an increased power saving. The level of power saving is software-controllable via the settings in the PCU\_PCONF<sub>n</sub> registers for power domain #2 onwards. The settings for power domains #0 and #1 can not be changed. Therefore, power domains #0 and #1 remain connected to the power supply for all modes beside STANDBY.

Figure 92 shows an example for a mode transition from RUN0 to HALT and back, which will result in power domain #2 being powered down during the HALT mode. In this case, PCU\_PCONF2.HALT is programmed to be '0'.

When the MC\_PCU receives the mode change request to HALT mode, it starts its power-down phase. During the power-down phase, clocks are disabled and the reset is asserted resulting in a loss of all information for this power domain.

Then the power domain is disconnected from the power supply (power-down state).

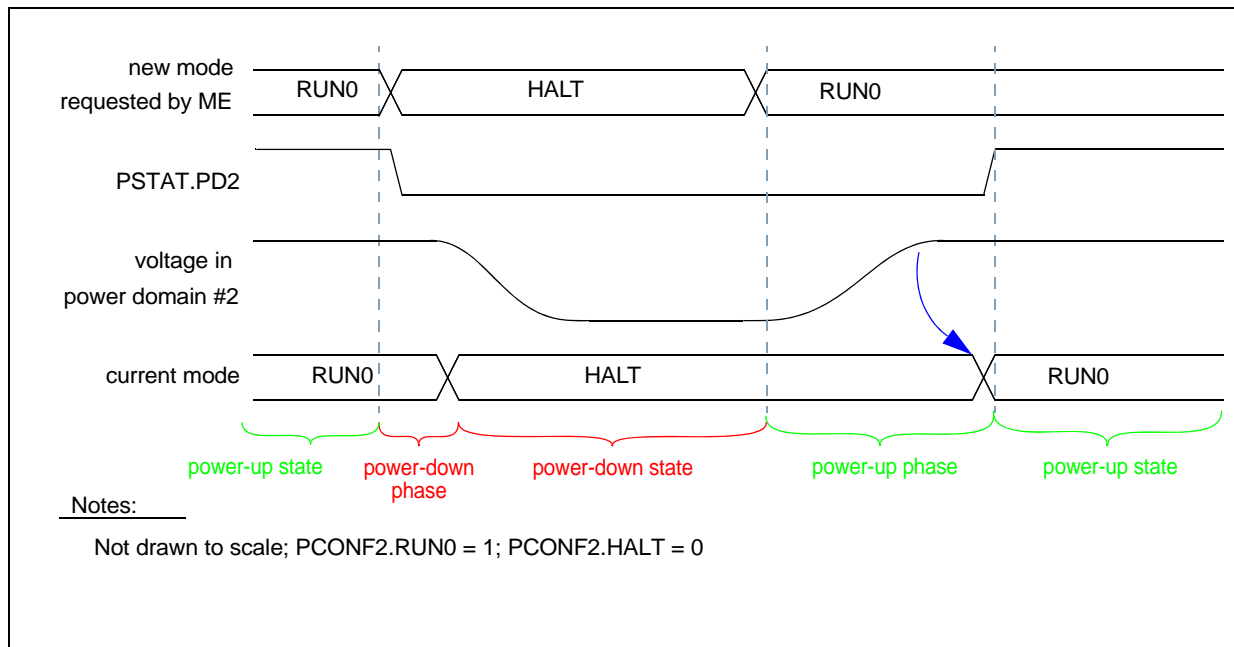


Figure 92. MC\_PCU Events During Power Sequences (non-STANDBY mode)

When the MC\_PCU receives a mode change request to RUN0, it starts its power-up phase if PCU\_PCONF2.RUN0 is '1'. The power domain is re-connected to the power supply, and the voltage in power domain #2 will increase slowly. Once the voltage of power domain #2 is within an operable range, its clocks are enabled, and its resets are deasserted (power-up state).

*Note: It is possible that, due to a mode change, power-up is requested before a power domain completed its power-down sequence. In this case, the information in that power domain is lost.*

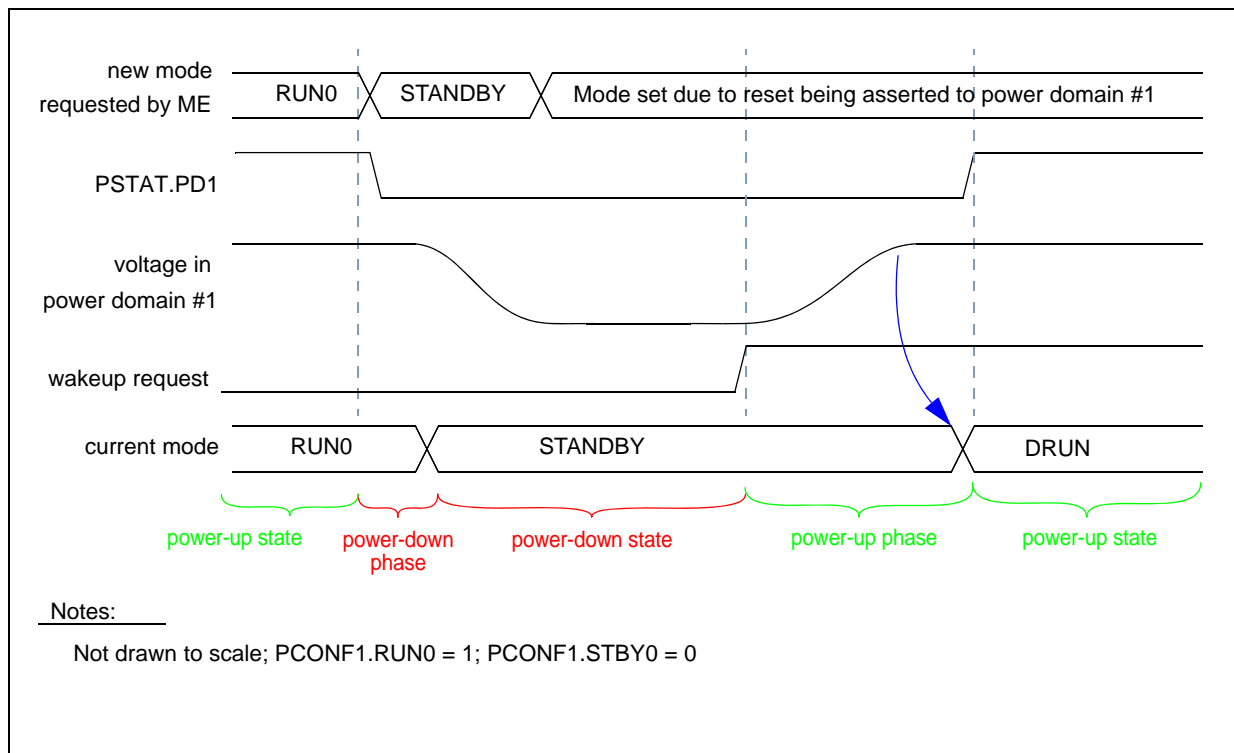
#### 10.4.4.2 STANDBY mode transition

STANDBY offers the maximum power saving. The level of power saving is software-controllable via the settings in the PCU\_PCONF<sub>n</sub> registers for power domain #2 onwards. Power domain #0 stays connected to the power supply while power domain #1 is disconnected from the power supply. Amongst others power domain #1 contains the platform and the MC\_ME. Therefore this mode differs from all other user/system modes.

Once STANDBY is entered it can only be left via a system wakeup. On exiting the STANDBY mode, all power domains are powered up according to the settings in the PCU\_PCONF<sub>n</sub> registers, and the DRUN mode is entered. In DRUN mode, at least power domains #0 and #1 are powered.

*Figure 93* shows an example for a mode transition from RUN0 to STANDBY to DRUN. All power domains which have PCU\_PCONF<sub>n</sub>.STBY0 cleared will enter power-down phase. In this example only power domain #1 will be disabled during STANDBY mode.

When the MC\_PCU receives the mode change request to STANDBY mode it starts the power down phase for power domain #1. During the power down phase, clocks are disabled and reset is asserted resulting in a loss of all information for this power domain. Then the power domain is disconnected from the power supply (power-down state).



**Figure 93. MC\_PCU Events During Power Sequences (STANDBY mode)**

When the MC\_PCU receives a system wakeup request, it starts the power-up phase. The power domain is re-connected to the power supply and the voltage in power domain #1 will increase slowly. Once the voltage is in an operable range, clocks are enabled and the reset is deasserted (power-up state).

*Note: It is possible that due to a wakeup request, power-up is requested before a power domain completed its power-down sequence. In this case, the information in that power domain is lost.*

**10.4.4.3 Power saving for memories during STANDBY mode**

All memories which are not powered down during STANDBY mode automatically enter a power saving state. No software configuration is required to enable this power saving state. While a memory is residing in this state an increased power saving is achieved. Data in the memories is retained.

**10.5 Initialization information**

To initialize the MC\_PCU, the registers PCU\_PCONF2...3 should be programmed. After programming is done, those registers should no longer be changed.

## 10.6 Application information

### 10.6.1 STANDBY Mode Considerations

STANDBY offers maximum power saving possibility. But power is only saved during the time a power domain is disconnected from the supply. Increased power is required when a power domain is re-connected to the power supply. Additional power is required during restoring the information (for example, in the platform).

Care should be taken that the time during which the SoC is operating in STANDBY mode is significantly longer than the required time for restoring the information.

# 11 Voltage Regulators and Power Supplies

## 11.1 Voltage regulators

The power blocks provide a 1.2 V digital supply to the internal logic of the device. The main supply is (3.3 V–5 V  $\pm$  10%) and digital/regulated output supply is (1.2 V  $\pm$  10%). The voltage regulator used in SPC560B54/6x comprises three regulators.

- High power regulator (HPREG)
- Low power regulator (LPREG)
- Ultra low power regulator (ULPREG)

The HPREG and LPREG regulators are switched off during STANDBY mode to save consumption from the regulator itself. In STANDBY mode, the supply is provided by the ULPREG regulator.

In STOP mode, the user can configure the HPREG regulator to switch-off (Refer to MC\_ME chapter). In this case, when current is low enough to be handled by LPREG alone, the HPREG regulator is switch-off and the supply is provided by the LPREG regulator.

The internal voltage regulator requires an external capacitance (CREG) to be connected to the device in order to provide a stable low voltage digital supply to the device. Capacitances should be placed on the board as near as possible to the associated pins.

The regulator has two digital domains, one for the high power regulator (HPREG) and the low power regulator (LPREG) called “High Power domain” and another one for the ultra low power regulator (ULPREG) called “Standby domain.” For each domain there is a low voltage detector for the 1.2 V output voltage. Additionally there are two low voltage detectors for the main/input supply with different thresholds, one at the 3.3 V level and the other one at the 5 V level.

### 11.1.1 High power regulator (HPREG)

The HPREG converts the 3.3 V–5 V input supply to a 1.2 V digital supply. For more information, see the voltage regulator electrical characteristics section of the datasheet.

The regulator can be switched off by software. Refer to the main voltage regulator control bit (MVRON) of the mode configuration registers in the mode entry module chapter of the reference manuals.

### 11.1.2 Low power regulator (LPREG)

The LPREG generates power for the device in the STOP mode, providing the output supply of 1.2 V. It always sees the minimum external capacitance. The control part of the regulator can be used to disable the low power regulator. It is managed by MC\_ME.

### 11.1.3 Ultra low power regulator (ULPREG)

The ULPREG generates power for the standby domain as well as a part of the main domain and might or might not see the external capacitance. The control circuit of ULPREG can be used to disable the ultra low power regulator by software: this action is managed by MC\_ME.



### 11.1.4 LVDs and POR

There are three kinds of LVD available:

1. LVD\_MAIN for the 3.3 V–5 V input supply with thresholds at approximately 3 V level<sup>(i)</sup>
2. LVD\_MAIN5 for the 3.3 V–5 V input supply with threshold at approximately 4.5 V level<sup>j</sup>
3. LVD\_DIG for the 1.2 V output voltage

The LVD\_MAIN and LVD\_MAIN5 sense the 3.3 V–5 V power supply for CORE, shared with IO ring supply and indicate when the 3.3 V–5 V supply is stabilized.

Two LVD\_DIGs are provided in the design. One LVD\_DIG is placed in the high power domain and senses the HPREG/LPREG output notifying that the 1.2 V output is stable. The other LVD\_DIG is placed in the standby domain and senses the standby 1.2 V supply level notifying that the 1.2 V output is stable. The reference voltage used for all LVDs is generated by the low power reference generator and is trimmed for LVD\_DIG, using the bits LP[4:7]. Therefore, during the pre-trimming period, LVD\_DIG exhibits higher thresholds, whereas during post trimming, the thresholds come in the desired range. Power-down pins are provided for LVDs. When LVDs are power-down, their outputs are pulled high.

POR is required to initialize the device during supply rise. POR works only on the rising edge of the main supply. To ensure its functioning during the following rising edge of the supply, it is reset by the output of the LVD\_MAIN block when main supply reaches below the lower voltage threshold of the LVD\_MAIN.

POR is asserted on power-up when V<sub>DD</sub> supply is above V<sub>PORUP</sub> min (refer to datasheet for details). It will be released only after V<sub>DD</sub> supply is above V<sub>PORH</sub> (refer to datasheet for details). V<sub>DD</sub> above V<sub>PORH</sub> ensures power management module including internal LVDs modules are fully functional.

### 11.1.5 VREG digital interface

The voltage regulator digital interface provides the temporization delay at initial power-up and at exit from low-power modes. A signal, indicating that Ultra Low Power domain is powered, is used at power-up to release reset to temporization counter. At exit from low-power modes, the power-down for high power regulator request signal is monitored by the digital interface and used to release reset to the temporization counter. In both cases, on completion of the delay counter, an end-of-count signal is released, it is gated with another signal indicating main domain voltage fine in order to release the VREGOK signal. This is used by MC\_RGM to release the reset to the device. It manages other specific requirements, like the transition between high power/low power mode to ultra low power mode avoiding a voltage drop below the permissible threshold limit of 1.08 V.

The VREG digital interface also holds control register to mask 5 V LVD status coming from the voltage regulator at the power-up.

### 11.1.6 Register description

The VREG\_CTL register is mapped to the MC\_PCU address space as described in [Chapter 10: Power Control Unit \(MC\\_PCU\)](#).

---

j. See section “Voltage monitor electrical characteristics” of the datasheet for detailed information about this voltage value.

Address: 0xC3FE\_8080 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5V_LVD_MASK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 94. Voltage Regulator Control Register (VREG\_CTL)

Table 95. VREG\_CTL field descriptions

Field	Description
5V_LVD_MASK	Mask bit for 5 V LVD from regulator This is a read/write bit and must be unmasked by writing a '1' by software to generate LVD functional reset request to MC_RGM for 5 V trip. 1: 5 V LVD is masked 0: 5 V LVD is not masked.

## 11.2 Power supply strategy

From a power-routing perspective, the device is organized as follows.

The device provides four dedicated supply domains at package level:

1. HV (high voltage external power supply for I/Os and most analog module) — This must be provided externally through VDD\_HV/VSS\_HV power pins. Voltage values should be aligned with  $V_{DD}/V_{SS}$ . Refer to datasheet for details.
2. ADC (high voltage external power supply for ADC module) — This must be provided externally through VDD\_HV\_ADC/VSS\_HV\_ADC power pins. Voltage values should be aligned with  $V_{DD\_HV\_ADC}/V_{SS\_HV\_ADC}$ . Refer to datasheet for details.
3. BV (high voltage external power supply for voltage regulator module) — This must be provided externally through VDD\_BV/VSS\_BV power pins. Voltage values should be aligned with  $V_{DD}/V_{SS}$ . Refer to datasheet for details.
4. LV (low voltage internal power supply for core, FMPLL and Flash digital logic) — This is generated internally by embedded voltage regulator and provided to the core, FMPLL and Flash. Three VDD\_LV/VSS\_LV pins pairs are provided to connect the three decoupling capacitances. This is generated internally by internal voltage regulator but provided outside to connect stability capacitor. Refer to datasheet for details.

The four dedicated supply domains are further divided within the package in order to reduce as much as possible EMC and noise issues.

- HV\_IO: High voltage pad supply
- HV\_FLAn: High voltage Flash supply
- HV\_OSC0REG<sup>(k)</sup>: High voltage external oscillator and regulator supply
- HV\_ADR: High voltage reference for ADC module. Supplies are further star routed to reduce impact of ADC resistive reference on ADC capacitive reference accuracy.
- HV\_ADV: High voltage supply for ADC module
- BV: High voltage supply for voltage regulator ballast. These two ballast pads are used to supply core and Flash. Each pad contains two ballasts to supply 80 mA and 20 mA respectively. Core is hence supplied through two ballasts of 80 mA capability and CFlash and DFlash through two 20 mA ballasts. The HV supply for both ballasts is shorted through double bonding.
- LV\_COR: Low voltage supply for the core. It is also used to provide supply for FMPLL through double bonding.
- LV\_FLAn: Low voltage supply for Flash module n. It is supplied with dedicated ballast and shorted to LV\_COR through double bonding.
- LV\_PLL<sup>(l)</sup>: Low voltage supply for FMPLL

### 11.3 Power domain organization

Based on stringent requirements for current consumption in different operational modes, the device is partitioned into different power domains. Organization into these power domains primarily means separate power supplies which are separated from each other by use of power switches (switch SW1 for power domain No. 1 and switch SW2 for power domain No. 2 as shown in [Figure 95](#)). These different separated power supplies are hence enabling to switch off power to certain regions of the device to avoid even leakage current consumption in logic supplied by the corresponding power supply.

This device employs three primary power domains, namely PD0, PD1 and PD2. As PCU supports dynamic power down of domains based on different device mode, such a possible domain is depicted below in dotted periphery.

Power domain organization and connections to the internal regulator are depicted in [Figure 95](#).

- 
- k. Regulator ground is separated from oscillator ground and shorted to the LV ground through star routing
  - l. During production test, it is also possible to provide the VDD\_LV externally through pins by configuring regulator in bypass mode.

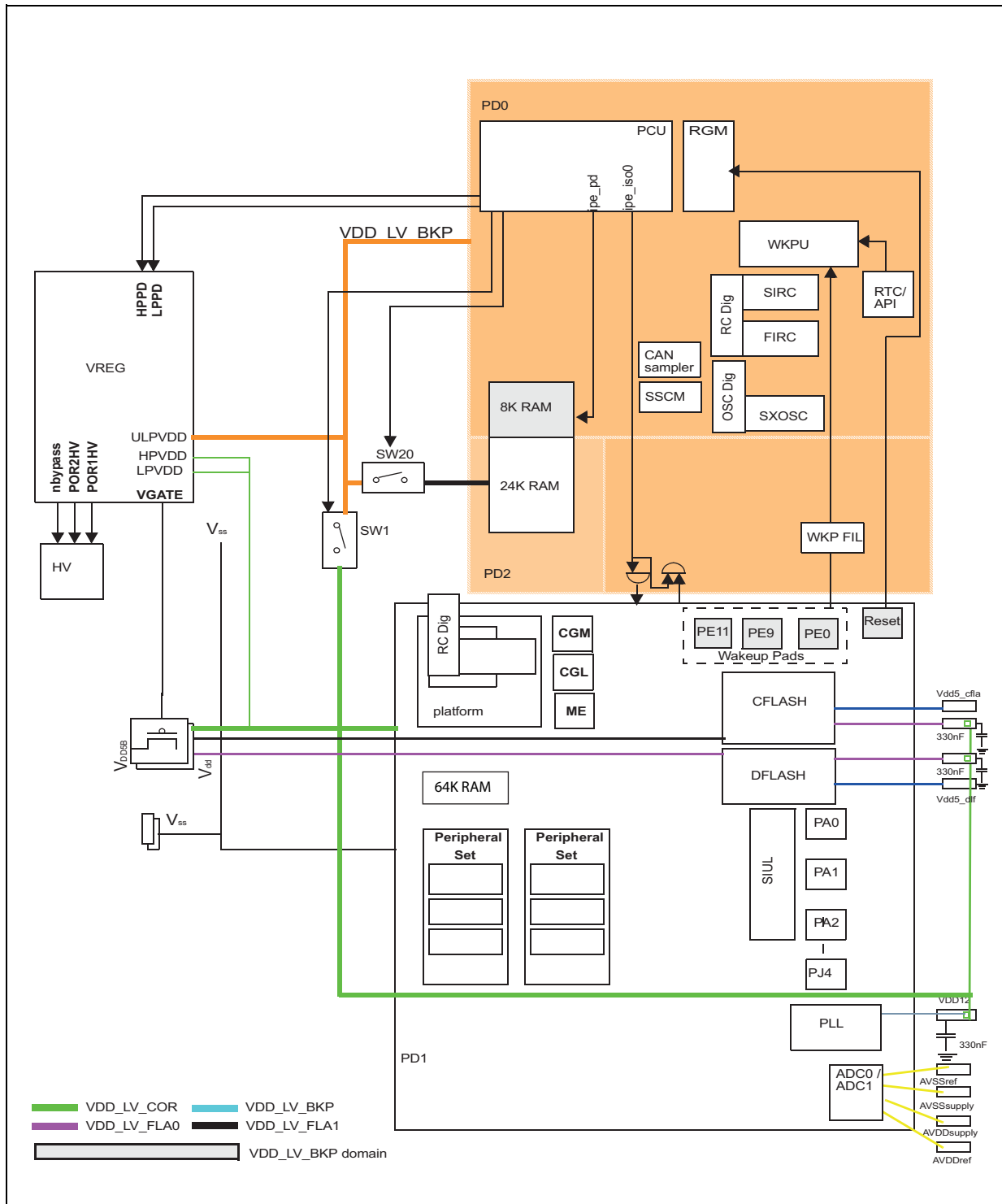


Figure 95. Power domain organization

## 12 Wakeup Unit (WKPU)

### 12.1 Overview

The Wakeup Unit supports 2 internal sources and up to 27 external sources that can generate interrupts or wakeup events, of which 1 can cause non-maskable interrupt requests or wakeup events. [Figure 96](#) is the block diagram of the Wakeup Unit and its interfaces to other system components.

The wakeup vector mapping is shown in [Table 96](#). All unused WKPU pins must use a pull resistor — either pullup (internal or external) or pulldown (external) — to ensure no leakage from floating inputs.

**Table 96. Wakeup vector mapping**

Wakeup number	Port	SIU PCR#	Port input function <sup>(1)</sup> (can be used in conjunction with WKPU function)	WKPU IRQ to INTC	IRQ#	WISR	Register <sup>(2)</sup> bit position	Package			
								100-pin QFP	144-pin QFP	176-pin QFP	208-pin BGA
WKUP0	API	n/a <sup>(3)</sup>	—	WakeUpIRQ0	46	EIF0	31	✓ <sup>(3)</sup>	✓ <sup>(3)</sup>	✓ <sup>(3)</sup>	✓ <sup>(3)</sup>
WKUP1	RTC	n/a <sup>(3)</sup>	—			EIF1	30	✓ <sup>(3)</sup>	✓ <sup>(3)</sup>	✓ <sup>(3)</sup>	✓ <sup>(3)</sup>
WKUP2	PA1	PCR1	NMI			EIF2	29	✓	✓	✓	✓
WKUP3	PA2	PCR2	—			EIF3	28	✓	✓	✓	✓
WKUP4	PB1	PCR17	CAN0RX, LIN0RX			EIF4	27	✓	✓	✓	✓
WKUP5	PC11	PCR43	CAN1RX, CAN4RX			EIF5	26	✓	✓	✓	✓
WKUP6	PE0	PCR64	CAN5RX			EIF6	25	✓	✓	✓	✓
WKUP7	PE9	PCR73	CAN2RX, CAN3RX			EIF7	24	✓	✓	✓	✓

Table 96. Wakeup vector mapping (continued)

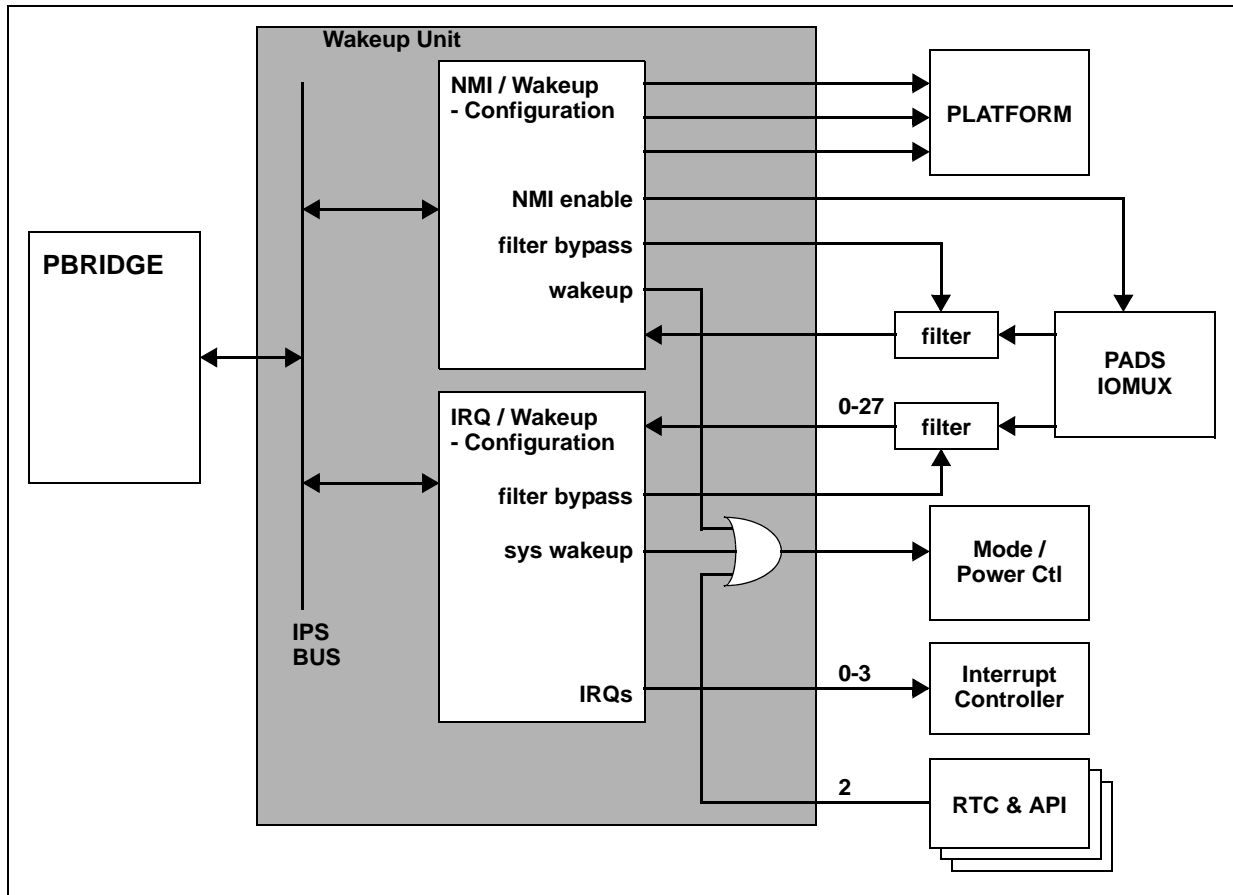
Wakeup number	Port	SIU PCR#	Port input function <sup>(1)</sup> (can be used in conjunction with WKPU function)	WKPU IRQ to INTC	IRQ#	WISR	Register <sup>(2)</sup> bit position	Package			
								100-pin QFP	144-pin QFP	176-pin QFP	208-pin BGA
WKUP8	PB10	PCR26	—	WakeUpIRQ1	47	EIF8	23	✓	✓	✓	✓
WKUP9	PA4	PCR4	LIN5RX			EIF9	22	✓	✓	✓	✓
WKUP1 0	PA15	PCR15	—			EIF10	21	✓	✓	✓	✓
WKUP1 1	PB3	PCR19	LIN0RX			EIF11	20	✓	✓	✓	✓
WKUP1 2	PC7	PCR39	LIN1RX			EIF12	19	✓	✓	✓	✓
WKUP1 3	PC9	PCR41	LIN2RX			EIF13	18	✓	✓	✓	✓
WKUP1 4	PE11	PCR75	LIN3RX			EIF14	17	✓	✓	✓	✓
WKUP1 5	PF11	PCR91	LIN4RX			EIF15	16	x <sup>(4)</sup>	✓	✓	✓
WKUP1 6	PF13	PCR93	LIN5RX			EIF16	15	x <sup>(4)</sup>	✓	✓	✓
WKUP1 7	PG3	PCR99	—	WakeUpIRQ2	48	EIF17	14	x <sup>(4)</sup>	✓	✓	✓
WKUP1 8	PG5	PCR10 1	—			EIF18	13	x <sup>(4)</sup>	✓	✓	✓
WKUP1 9	PA0	PCR0	—			EIF19	12	✓	✓	✓	✓
WKUP2 0	PG7	PCR10 3	LIN6RX			EIF20	11	x <sup>(4)</sup>	✓	✓	✓
WKUP2 1	PG9	PCR10 5	LIN7RX			EIF21	10	x <sup>(4)</sup>	✓	✓	✓
WKUP2 2	PF9	PCR89	CAN2RX, CAN3RX			EIF22	9	x <sup>(4)</sup>	✓	✓	✓
WKUP2 3	PI3	PCR13 1	LIN9RX			EIF23	8	x <sup>(4)</sup>	x <sup>(4)</sup>	✓	✓

Table 96. Wakeup vector mapping (continued)

Wakeup number	Port	SIU PCR#	Port input function <sup>(1)</sup> (can be used in conjunction with WKPU function)	WKPU IRQ to INTC	IRQ#	WISR	Register <sup>(2)</sup> bit position	Package			
								100-pin QFP	144-pin QFP	176-pin QFP	208-pin BGA
WKUP2 4	PI1	PCR12 9	LIN8RX	WakeUpIRQ3	49	EIF24	7	x <sup>(4)</sup>	x <sup>(4)</sup>	✓	✓
WKUP2 5	PB8	PCR24	—			EIF25	6	✓	✓	✓	✓
WKUP2 6	PB9	PCR25	—			EIF26	5	✓	✓	✓	✓
WKUP2 7	PD0	PCR48	—			EIF27	4	✓	✓	✓	✓
WKUP2 8	PD1	PCR49	—			EIF28	3	✓	✓	✓	✓

1. This column does not contain an exhaustive list of functions on that pin. Rather, it includes peripheral communication functions (such as CAN and LINFlex Rx) that could be used to wake up the microcontroller. DSPI pins are not included because DSPI would typically be used in master mode.
2. WISR, IRER, WRER, WIFEER, WIFEER, WIFER, WIPUER.
3. Port not required to use timer functions.
4. Unavailable WKPU pins must use internal pullup enabled using WIPUER.

Figure 96. WKPU block diagram



## 12.2 Features

The Wakeup Unit supports these distinctive features:

- Non-maskable interrupt support with
  - 1 NMI source with bypassable glitch filter
  - Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
  - Edge detection
- External wakeup/interrupt support with
  - 4 system interrupt vectors for up to 29 interrupt sources
  - Analog glitch filter per each wakeup line
  - Independent interrupt mask
  - Edge detection
  - Configurable system wakeup triggering from all interrupt sources
  - Configurable pullup
- On-chip wakeup support
  - 2 wakeup sources
  - Wakeup status mapped to same register as external wakeup/interrupt status



## 12.3 External signal description

The Wakeup Unit has 29 signal inputs that can be used as external interrupt sources in normal RUN mode or as system wakeup sources in all power down modes.

The 27 external signal inputs include one signal input that can be used as a non-maskable interrupt source in normal RUN, HALT or STOP modes or a system wakeup source in STOP or STANDBY modes. The exception is with ports PB[8] and PB[9], which have wakeup functionality in all modes except STANDBY.

*Note: The user should be aware that the Wake-up pins are enabled in ALL modes, therefore, the Wake-up pins should be correctly terminated to ensure minimal current consumption. Any unused Wake-up signal input should be terminated by using an external pull-up or pull-down, or by internal pull-up enabled at WKPU\_WIPUER. Also, care has to be taken on packages where the Wake-up signal inputs are not bonded. For these packages the user must ensure the internal pull-up are enabled for those signals not bonded.*

## 12.4 Memory map and register description

This section provides a detailed description of all registers accessible in the WKPU module.

### 12.4.1 Memory map

*Table 97* gives an overview on the WKPU registers implemented.

**Table 97. WKPU memory map**

Base address: 0xC3F9_4000		
Address offset	Register name	Location
0x00	NMI Status Flag Register (NSR)	<a href="#">on page 274</a>
0x04 – 0x07	Reserved	
0x08	NMI Configuration Register (NCR)	<a href="#">on page 274</a>
0x0C – 0x13	Reserved	
0x14	Wakeup/Interrupt Status Flag Register (WISR)	<a href="#">on page 276</a>
0x18	Interrupt Request Enable Register (IRER)	<a href="#">on page 276</a>
0x1C	Wakeup Request Enable Register (WRER)	<a href="#">on page 277</a>
0x20 – 0x27	Reserved	
0x28	Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)	<a href="#">on page 277</a>
0x2C	Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)	<a href="#">on page 277</a>
0x30	Wakeup/Interrupt Filter Enable Register (WIFER)	<a href="#">on page 278</a>
0x34	Wakeup/Interrupt Pullup Enable Register (WIPUER)	<a href="#">on page 278</a>

*Note: Reserved registers will read as 0, writes will have no effect. If SSCM\_ERROR[RAE] is enabled, a transfer error will be issued when trying to access completely reserved register space.*

### 12.4.2 NMI Status Flag Register (NSR)

This register holds the non-maskable interrupt status flags.

Offset: 0x00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NIF0	NOVF0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c	w1c														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 97. NMI Status Flag Register (NSR)

Table 98. NSR field descriptions

Field	Description
NIF0	<p>NMI Status Flag</p> <p>If enabled (NREE0 or NFEE0 set), NIF0 causes an interrupt request.</p> <p>1 An event as defined by NREE0 and NFEE0 has occurred</p> <p>0 No event has occurred on the pad</p>
NOVF0	<p>NMI Overrun Status Flag</p> <p>It will be a copy of the current NIF0 value whenever an NMI event occurs, thereby indicating to the software that an NMI occurred while the last one was not yet serviced. If enabled (NREE0 or NFEE0 set), NOVF0 causes an interrupt request.</p> <p>1 An overrun has occurred on NMI input</p> <p>0 No overrun has occurred on NMI input</p>

### 12.4.3 NMI Configuration Register (NCR)

This register holds the configuration bits for the non-maskable interrupt settings.

Offset: 0x08

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NLOCK0	NDSS0		NWRE0	0	NREE0	NFEE0	NFE0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 98. NMI Configuration Register (NCR)

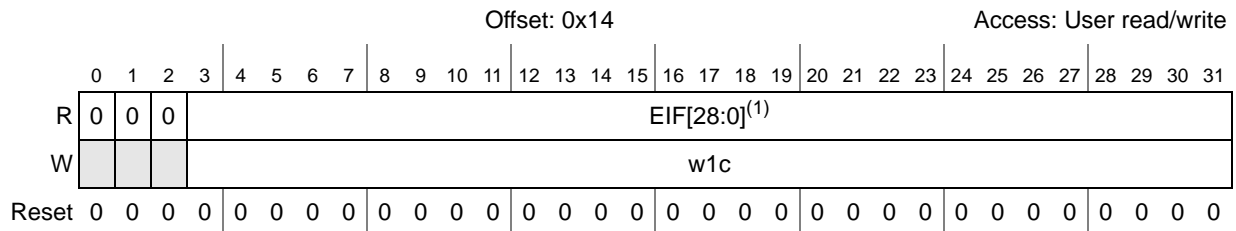
Table 99. NCR field descriptions

Field	Description
NLOCK0	NMI Configuration Lock Register Writing a 1 to this bit locks the configuration for the NMI until it is unlocked by a system reset. Writing a 0 has no effect.
NDSS0	NMI Destination Source Select 00 Non-maskable interrupt 01 Critical interrupt 10 Machine check request 11 Reserved—no NMI, critical interrupt, or machine check request generated
NWRE0	NMI Wakeup Request Enable 1 A set NIF0 bit or set NOVFO bit causes a system wakeup request 0 System wakeup requests from the corresponding NIF0 bit are disabled <b>Note:</b> Software should only enable the NMI after the IVPR/IVOR registers have been configured. This should be noted when booting from RESET or STANDBY mode as all registers will have been cleared to their reset state.
NREE0	NMI Rising-edge Events Enable 1 Rising-edge event is enabled 0 Rising-edge event is disabled
NFEE0	NMI Falling-edge Events Enable 1 Falling-edge event is enabled 0 Falling-edge event is disabled
NFE0	NMI Filter Enable Enable analog glitch filter on the NMI pad input. 1 Filter is enabled 0 Filter is disabled

Note: Writing a '0' to both NREE0 and NFEE0 disables the NMI functionality completely (that is, no system wakeup or interrupt will be generated on any pad activity)!

### 12.4.4 Wakeup/Interrupt Status Flag Register (WISR)

This register holds the wakeup/interrupt flags.



**Figure 99. Wakeup/Interrupt Status Flag Register (WISR)**

1. EIF[24:20] and EIF[18:15] not available in 100-pin LQFP; EIF[24:23] not available in 144-pin LQFP

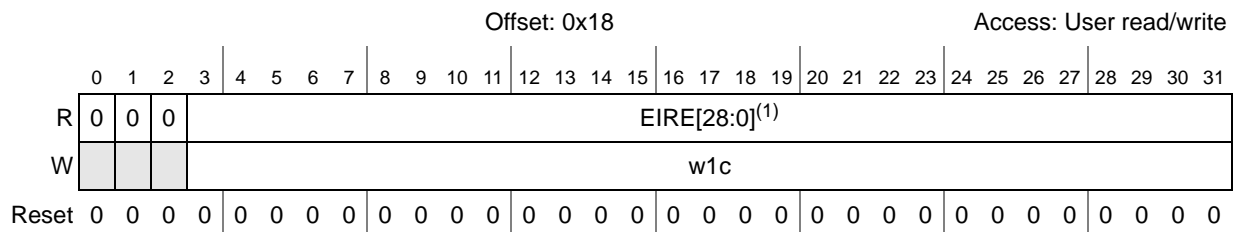
**Table 100. WISR field descriptions**

Field	Description
EIF[x]	External Wakeup/Interrupt WKPU[x] Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER[x]), EIF[x] causes an interrupt request. 1 An event as defined by WIREER and WIFEER has occurred 0 No event has occurred on the pad

*Note:* Status bits associated with on-chip wakeup sources are located to the left of the external wakeup/interrupt status bits and are read only. The wakeup for these sources must be configured and cleared at the on-chip wakeup source. Also, the configuration registers for the external interrupts/wakeups do not have corresponding bits.

### 12.4.5 Interrupt Request Enable Register (IRER)

This register is used to enable the interrupt messaging from the wakeup/interrupt pads to the interrupt controller.



**Figure 100. Interrupt Request Enable Register (IRER)**

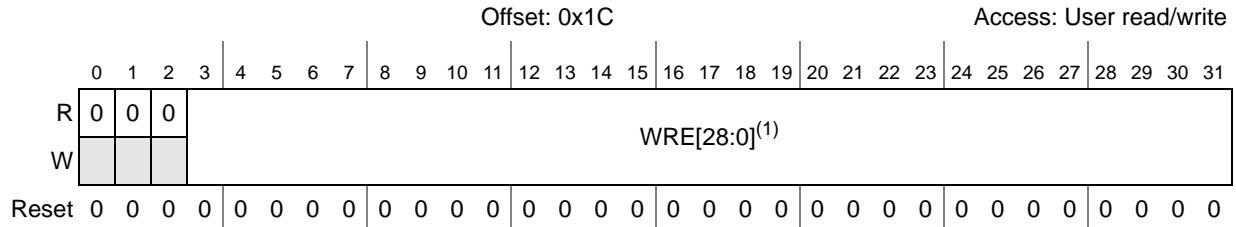
1. EIRE[24:20] and EIRE[18:15] not available in 100-pin LQFP; EIRE[24:23] not available in 144-pin LQFP

**Table 101. IRER field descriptions**

Field	Description
EIRE[x]	External Interrupt Request Enable x 1 A set EIF[x] bit causes an interrupt request 0 Interrupt requests from the corresponding EIF[x] bit are disabled

### 12.4.6 Wakeup Request Enable Register (WRER)

This register is used to enable the system wakeup messaging from the wakeup/interrupt pads to the mode entry and power control modules.



**Figure 101. Wakeup Request Enable Register (WRER)**

1. WRE[24:20] and WRE[18:15] not available in 100-pin LQFP; WRE[24:23] not available in 144-pin LQFP

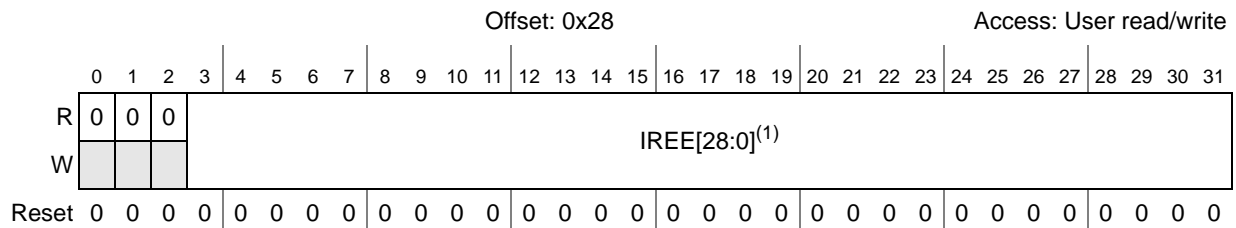
**Table 102. WRER field descriptions**

Field	Description
WRE[x]	External Wakeup Request Enable x 1 A set EIF[x] bit causes a system wakeup request 0 System wakeup requests from the corresponding EIF[x] bit are disabled

### 12.4.7 Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)

This register is used to enable rising-edge triggered events on the corresponding wakeup/interrupt pads.

*Note:* The RTC\_API can only be configured on the rising edge.



**Figure 102. Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)**

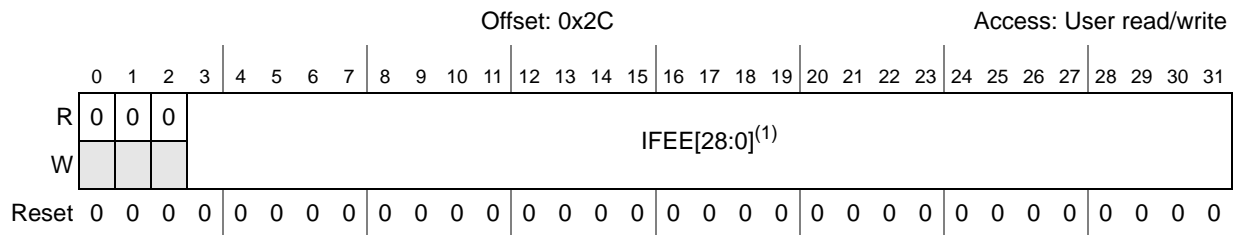
1. IREE[24:20] and IREE[18:15] not available in 100-pin LQFP; IREE[24:23] not available in 144-pin LQFP

**Table 103. WIREER field descriptions**

Field	Description
IREE[x]	External Interrupt Rising-edge Events Enable x 1 Rising-edge event is enabled 0 Rising-edge event is disabled

### 12.4.8 Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)

This register is used to enable falling-edge triggered events on the corresponding wakeup/interrupt pads.



**Figure 103. Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)**

1. IFEE[24:20] and IFEE[18:15] not available in 100-pin LQFP; IFEE[24:23] not available in 144-pin LQFP

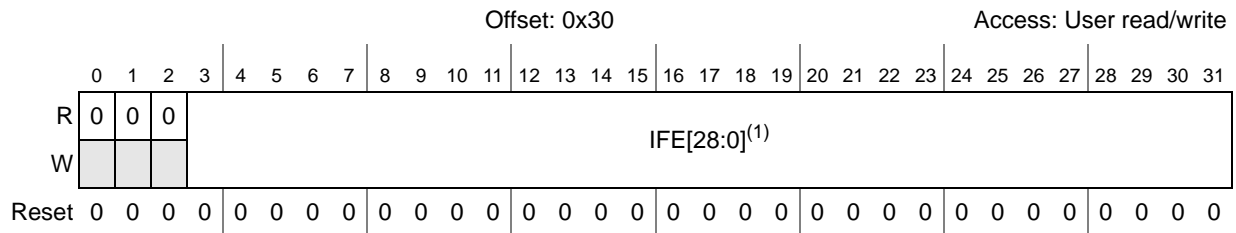
**Table 104. WIFEER field descriptions**

Field	Description
IFEE <sub>x</sub>	External Interrupt Falling-edge Events Enable x 1 Falling-edge event is enabled 0 Falling-edge event is disabled

### 12.4.9 Wakeup/Interrupt Filter Enable Register (WIFER)

This register is used to enable an analog filter on the corresponding interrupt pads to filter out glitches on the inputs.

*Note:* There is no analog filter for the RTC\_API.



**Figure 104. Wakeup/Interrupt Filter Enable Register (WIFER)**

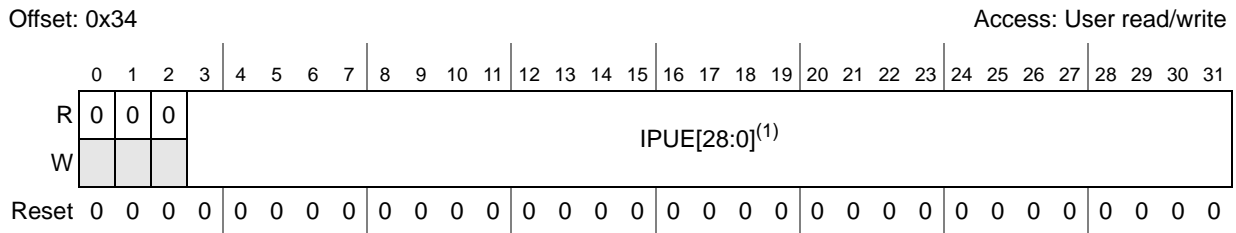
1. IFE[24:20] and IFE[18:15] not available in 100-pin LQFP; IFE[24:23] not available in 144-pin LQFP

**Table 105. WIFER field descriptions**

Field	Description
IFE[x]	External Interrupt Filter Enable x Enable analog glitch filter on the external interrupt pad input. 1 Filter is enabled 0 Filter is disabled

### 12.4.10 Wakeup/Interrupt Pullup Enable Register (WIPUER)

This register is used to enable a pullup on the corresponding interrupt pads to pull an unconnected wakeup/interrupt input to a value of '1'.



**Figure 105. Wakeup/Interrupt Pullup Enable Register (WIPUER)**

1. IPUE[24:20] and IPUE[18:15] not available in 100-pin LQFP; IPUE[24:23] not available in 144-pin LQFP

**Table 106. WIPUER field descriptions**

Field	Description
IPUE[x]	External Interrupt Pullup Enable x 1 Pullup is enabled 0 Pullup is disabled

## 12.5 Functional description

### 12.5.1 General

This section provides a complete functional description of the Wakeup Unit.

### 12.5.2 Non-maskable interrupts

The Wakeup Unit supports one non-maskable interrupt which is allocated to the following pins:

- 100-pin LQFP: Pin 7
- 144-pin LQFP: Pin 11
- 176-pin LQFP: Pin 19
- 208-pin BGA: Pin F3

The Wakeup Unit supports the generation of three types of interrupts from the NMI. The Wakeup Unit supports the capturing of a second event per NMI input before the interrupt is cleared, thus reducing the chance of losing an NMI event.

Each NMI passes through a bypassable analog glitch filter.

*Note:* Glitch filter control and pad configuration should be done while the NMI is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.

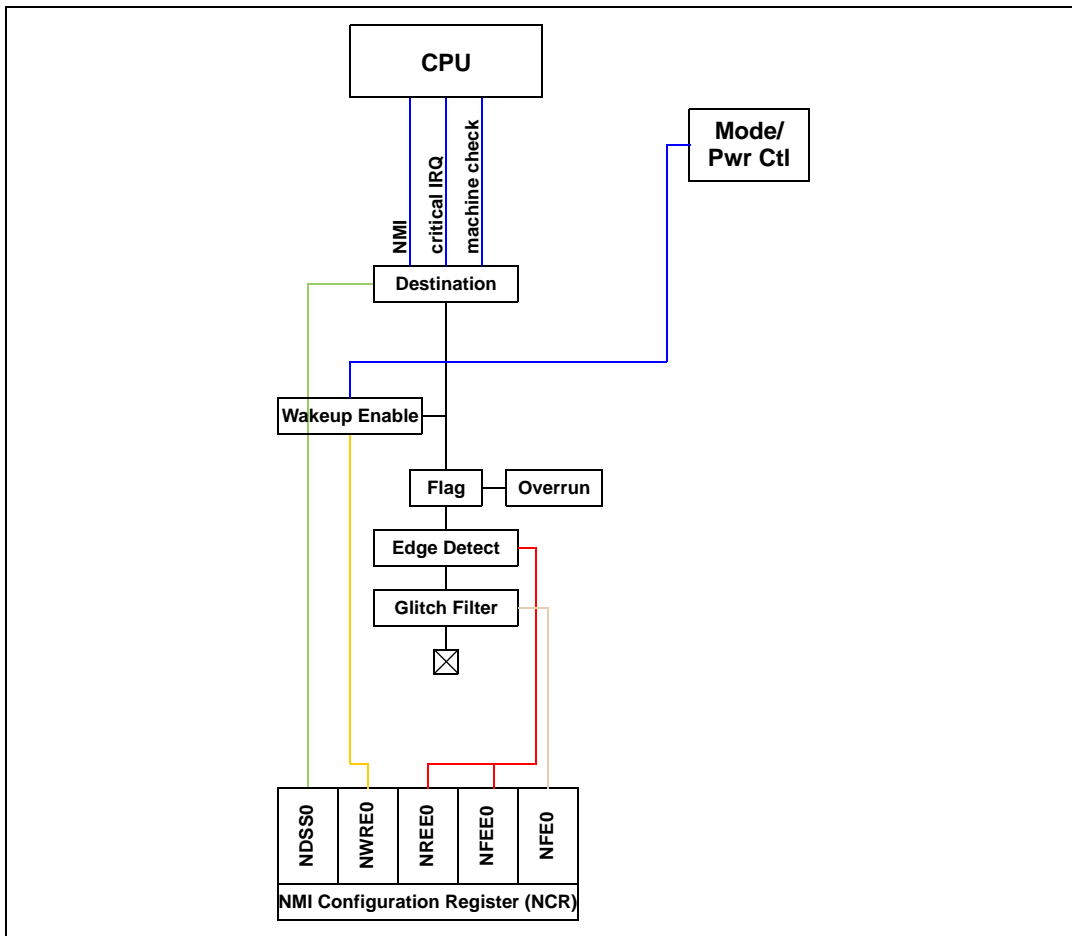


Figure 106. NMI pad diagram

### 12.5.2.1 NMI management

The NMI can be enabled or disabled using the single NCR register laid out to contain all configuration bits for an NMI in a single byte (see [Figure 98](#)). The pad defined as an NMI can be configured by the user to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled results in no interrupt being detected and should not be configured.

The active NMI edge is controlled by the user through the configuration of the NREE0 and NFEE0 bits.

*Note:* After reset, NREE0 and NFEE0 are set to '0', therefore the NMI functionality is disabled after reset and must be enabled explicitly by software.

Once the pad's NMI functionality has been enabled, the pad cannot be reconfigured in the IOMUX to override or disable the NMI.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS0 field. See [Table 99](#) for details.

An NMI supports a status flag and an overrun flag which are located in the NSR register (see [Figure 97](#)). The NIF0 and NOVFO fields in this register are cleared by writing a '1' to them; this prevents inadvertent overwriting of other flags in the register. The status flag is set



whenever an NMI event is detected. The overrun flag is set whenever an NMI event is detected and the status flag is set (that is, has not yet been cleared).

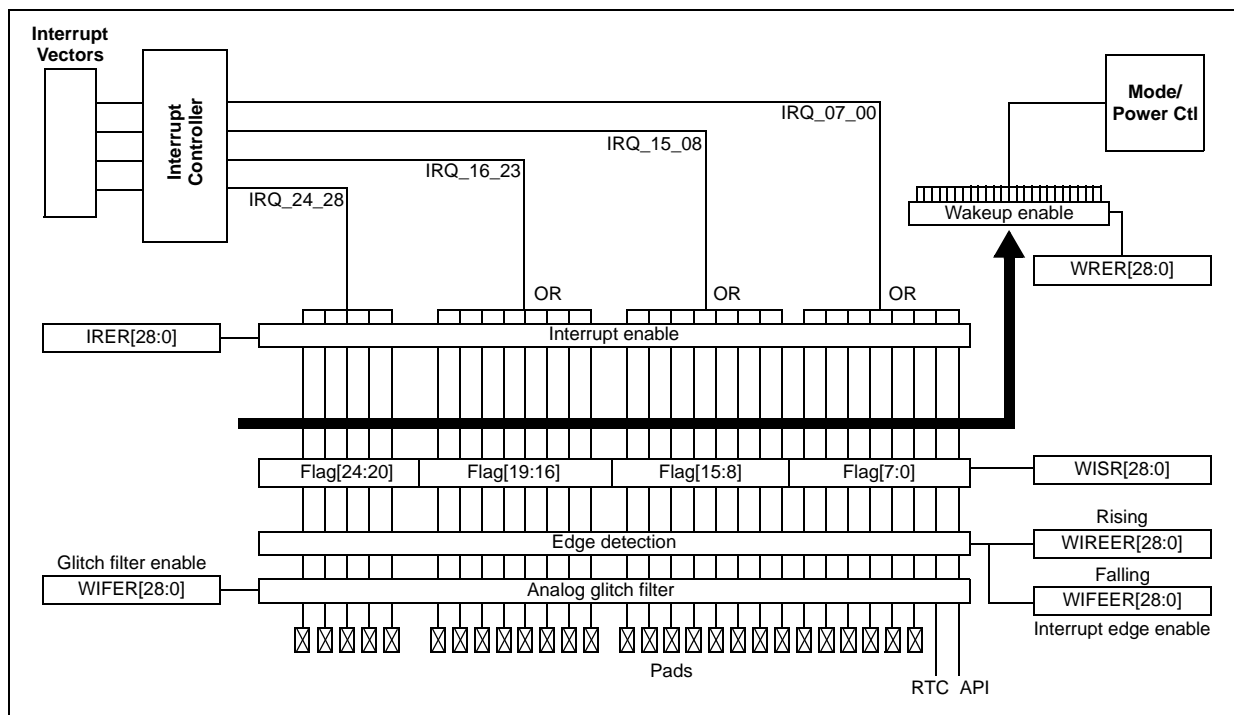
*Note: The overrun flag is cleared by writing a '1' to the appropriate overrun bit in the NSR register. If the status bit is cleared and the overrun bit is still set, the pending interrupt will not be cleared.*

### 12.5.3 External wakeups/interrupts

The Wakeup Unit supports up to 27 external wakeup/interrupts which can be allocated to any pad necessary at the SoC level. This allocation is fixed per SoC.

The Wakeup Unit supports up to four interrupt vectors to the interrupt controller of the SoC. Each interrupt vector can support up to the number of external interrupt sources from the device pads with the total across all vectors being equal to the number of external interrupt sources. Each external interrupt source is assigned to exactly one interrupt vector. The interrupt vector assignment is sequential so that one interrupt vector is for external interrupt sources 0 through N-1, the next is for N through N+M-1, and so forth.

See [Figure 107](#) for an overview of the external interrupt implementation for the example of four interrupt vectors with up to eight external interrupt sources each.



**Figure 107. External interrupt pad diagram**

All of the external interrupt pads within a single group have equal priority. It is the responsibility of the user software to search through the group of sources in the most appropriate way for their application.

*Note: Glitch filter control and pad configuration should be done while the external interrupt line is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.*

### 12.5.3.1 External interrupt management

Each external interrupt can be enabled or disabled independently. This can be performed using a single rolled up register ([Figure 100](#)). A pad defined as an external interrupt can be configured by the user to recognize external interrupts with an active rising edge, an active falling edge or both edges being active.

*Note: Writing a '0' to both IREE[x] and IFEE[x] disables the external interrupt functionality for that pad completely (that is, no system wakeup or interrupt will be generated on any activity on that pad)!*

The active IRQ edge is controlled by the users through the configuration of the registers WIREER and WIFEER.

Each external interrupt supports an individual flag which is held in the flag register (WISR). The bits in the WISR[EIF] field are cleared by writing a '1' to them; this prevents inadvertent overwriting of other flags in the register.

### 12.5.4 On-chip wakeups

The Wakeup Unit supports two on-chip wakeup sources. It combines the on-chip wakeups with the external ones to generate a single wakeup to the system.

#### 12.5.4.1 On-chip wakeup management

In order to allow software to determine the wakeup source at one location, on-chip wakeups are reported along with external wakeups in the WISR register (see [Figure 99](#) for details). Enabling and clearing of these wakeups are done via the on-chip wakeup source's own registers.

## 13 Real Time Clock / Autonomous Periodic Interrupt (RTC/API)

### 13.1 Overview

The RTC/API is a free running counter used for time keeping applications. The RTC may be configured to generate an interrupt at a predefined interval independent of the mode of operation (run mode or low power mode). If in a low power mode when the RTC interval is reached, the RTC first generates a wakeup and then assert the interrupt request. The RTC also supports an autonomous periodic interrupt (API) function used to generate a periodic wakeup request to exit a low power mode or an interrupt request.

### 13.2 Features

Features of the RTC/API include:

- 3 selectable counter clock sources
  - SIRC (128 kHz)
  - SXOSC (32 KHz)
  - FIRC (16 MHz)
- Optional 512 prescaler and optional 32 prescaler
- 32-bit counter
  - Supports times up to 1.5 months with 1 ms resolution
  - Runs in all modes of operation
  - Reset when disabled by software and by POR
- 12-bit compare value to support interrupt intervals of 1 s up to greater than 1 hr with 1 s resolution
- RTC compare value changeable while counter is running
- RTC status and control register are reset only by POR
- Autonomous periodic interrupt (API)
  - 10-bit compare value to support wakeup intervals of 1.0 ms to 1 s
  - Compare value changeable while counter is running
- Configurable interrupt for RTC match, API match, and RTC rollover
- Configurable wakeup event for RTC match, API match, and RTC rollover

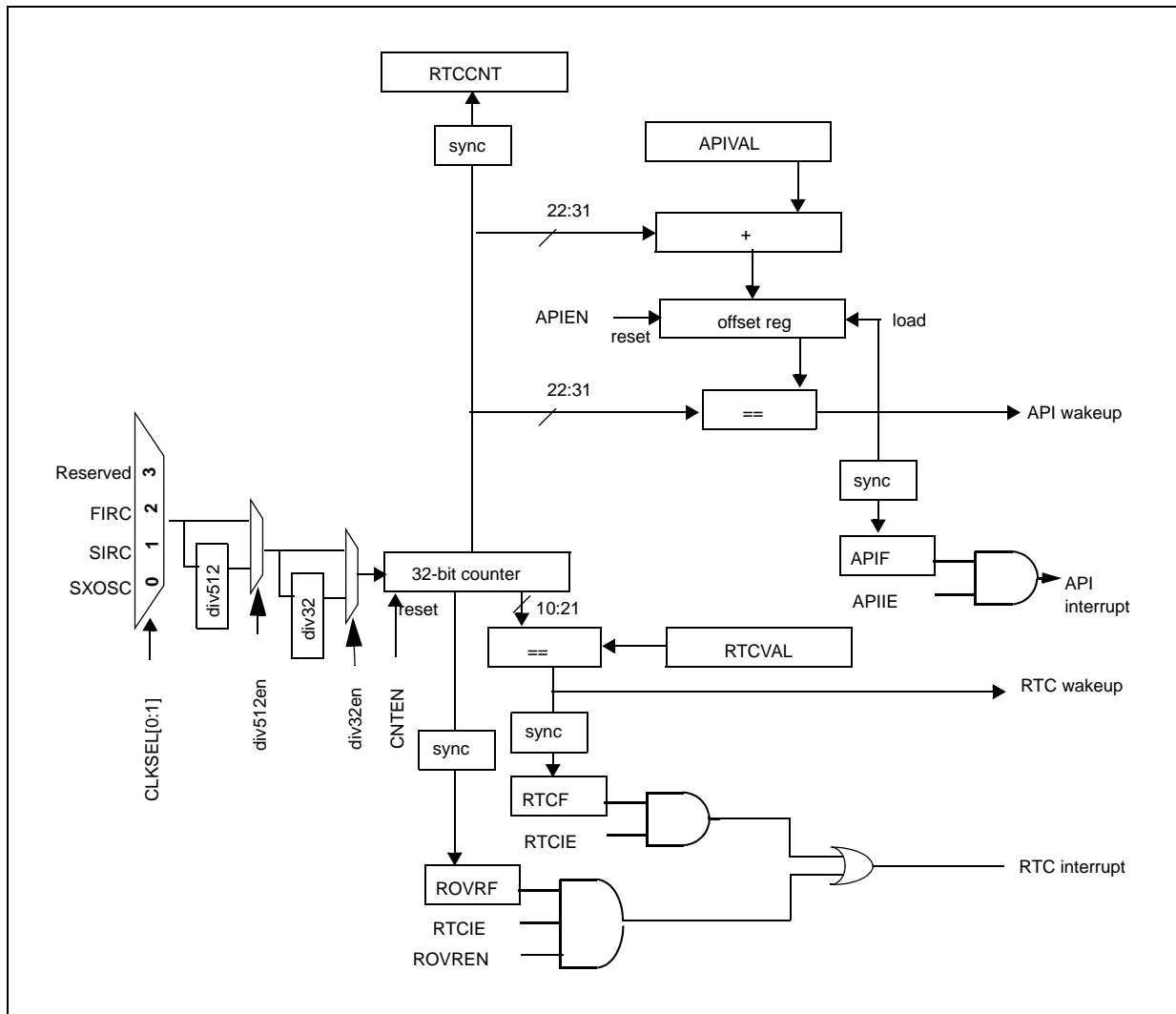


Figure 108. RTC/API block diagram

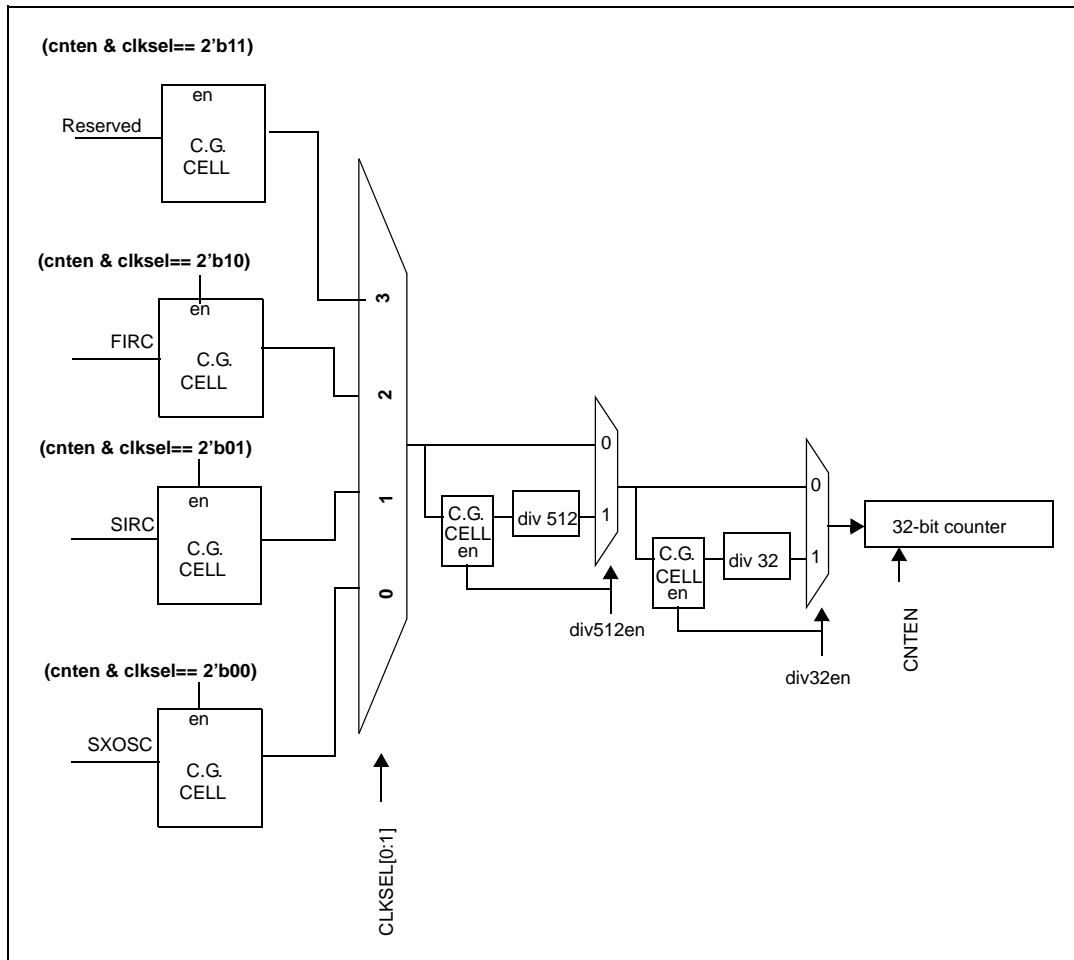


Figure 109. Clock gating for RTC clocks

### 13.3 Device-specific information

For SPC560B54/6x, the device specific information is the following:

- SXOSC, FIRC and SIRC clocks are provided as counter clocks for the RTC. Default clock on reset is SIRC divided by 4.
- The RTC will be reset on destructive reset, with the exception of software watchdog reset.
- The RTC provides a configurable divider by 512 to be optionally used when FIRC source is selected.

### 13.4 Modes of operation

#### 13.4.1 Functional mode

There are two functional modes of operation for the RTC: normal operation and low power mode. In normal operation, all RTC registers can read or written and the input isolation is

disabled. The RTC/API and associated interrupts are optionally enabled. In low power mode, the bus interface is disabled and the input isolation is enabled. The RTC/API is enabled if enabled prior to entry into low power mode.

### 13.4.2 Debug mode

On entering into the debug mode the RTC counter freezes on the last valid count if the RTCC[FRZEN] is set. On exit from debug mode counter continues from the frozen value.

## 13.5 Register descriptions

Table 107 lists the RTC/API registers.

Table 107. RTC/API register map

Base address: 0xC3FE_C000		
Address offset	Register	Location
0x0	RTC Supervisor Control Register (RTCSUPV)	on page 286
0x4	RTC Control Register (RTCC)	on page 287
0x8	RTC Status Register (RTCS)	on page 289
0xC	RTC Counter Register (RTCCNT)	on page 290

### 13.5.1 RTC Supervisor Control Register (RTCSUPV)

The RTCSUPV register contains the SUPV bit which determines whether other registers are accessible in supervisor mode or user mode.

Note: RTCSUPV register is accessible only in supervisor mode.

Offset: 0x0

Access: Read/write

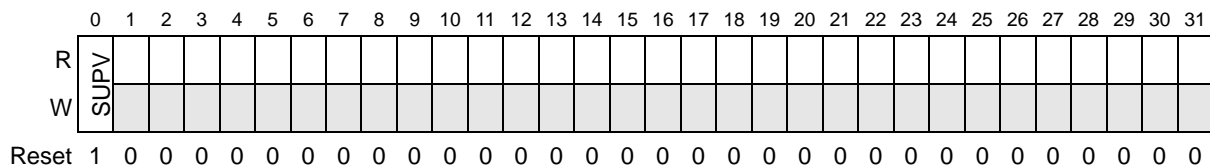


Figure 110. RTC Supervisor Control Register (RTCSUPV)

Table 108. RTCSUPV field descriptions

Field	Description
SUPV	RTC Supervisor Bit 0 All registers are accessible in both user as well as supervisor mode. 1 All other registers are accessible in supervisor mode only.

### 13.5.2 RTC Control Register (RTCC)

The RTCC register contains:

- RTC counter enable
- RTC interrupt enable
- RTC clock source select
- RTC compare value
- API enable
- API interrupt enable
- API compare value

Offset: 0x4

Access: User read/write

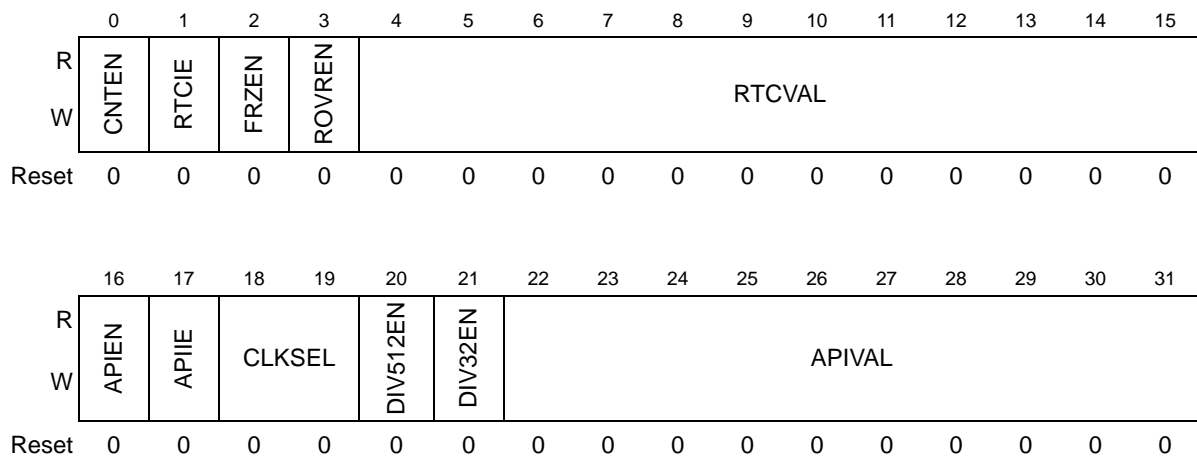


Figure 111. RTC Control Register (RTCC)

Table 109. RTCC field descriptions

Field	Description
CNTEN	Counter Enable The CNTEN field enables the RTC counter. Making CNTEN bit 1'b0 has the effect of asynchronously resetting (synchronous reset negation) all the RTC and API logic. This allows for the RTC configuration and clock source selection to be updated without causing synchronization issues. 1 Counter enabled 0 Counter disabled
RTCIE	RTC Interrupt Enable The RTCIE field enables interrupts requests to the system if RTCF is asserted. 1 RTC interrupts enabled 0 RTC interrupts disabled
FRZEN	Freeze Enable The counter freezes on entering the debug mode on the last valid count value if the FRZEN bit is set. After coming out of the debug mode, the counter starts from the frozen value. 0 Counter does not freeze in debug mode. 1 Counter freezes in debug mode.

Table 109. RTCC field descriptions (continued)

Field	Description
ROVREN	Counter Roll Over Wakeup/Interrupt Enable The ROVREN bit enables wakeup and interrupt requests when the RTC has rolled over from 0xFFFF_FFFF to 0x0000_0000. The RTCIE bit must also be set in order to generate an interrupt from a counter rollover. 1 RTC rollover wakeup/interrupt enabled 0 RTC rollover wakeup/interrupt disabled
RTCVAL	Note: RTC Compare Value The RTCVAL bits are compared to bits 10:21 of the RTC counter and if match sets RTCF. RTCVAL can be updated when the counter is running.
APIEN	Autonomous Periodic Interrupt Enable The APIEN bit enables the autonomous periodic interrupt function. 1 API enabled 0 API disabled
APIIE	API Interrupt Enable The APIIE bit enables interrupts requests to the system if APIF is asserted. 1 API interrupts enabled 0 API interrupts disabled
CLKSEL	Clock Select This field selects the clock source for the RTC. CLKSEL may only be updated when CNTEN is 0. The user should ensure that oscillator is enabled before selecting it as a clock source for RTC. 00 SXOSC 01 SIRC 10 FIRC 11 Reserved
DIV512EN	Divide by 512 enable The DIV512EN bit enables the 512 clock divider. DIV512EN may only be updated when CNTEN is 0. 0 Divide by 512 is disabled. 1 Divide by 512 is enabled.
DIV32EN	Divide by 32 enable The DIV32EN bit enables the 32 clock divider. DIV32EN may only be updated when CNTEN is 0. 0 Divide by 32 is disabled. 1 Divide by 32 is enabled.
APIVAL	API Compare Value The APIVAL field is compared with bits 22:31 of the RTC counter and if match asserts an interrupt/wakeup request. APIVAL may only be updated when APIEN is 0 or API function is undefined. Note: API functionality starts only when APIVAL is nonzero. The first API interrupt takes two more cycles because of synchronization of APIVAL to the RTC clock, and APIVAL + 1 cycles for subsequent occurrences. After that, interrupts are periodic in nature. Because of synchronization issues, the minimum supported value of APIVAL is 4.



### 13.5.3 RTC Status Register (RTCS)

The RTCS register contains:

- RTC interrupt flag
- API interrupt flag
- ROLLOVR Flag

Offset: 0x8

Access: User read/write

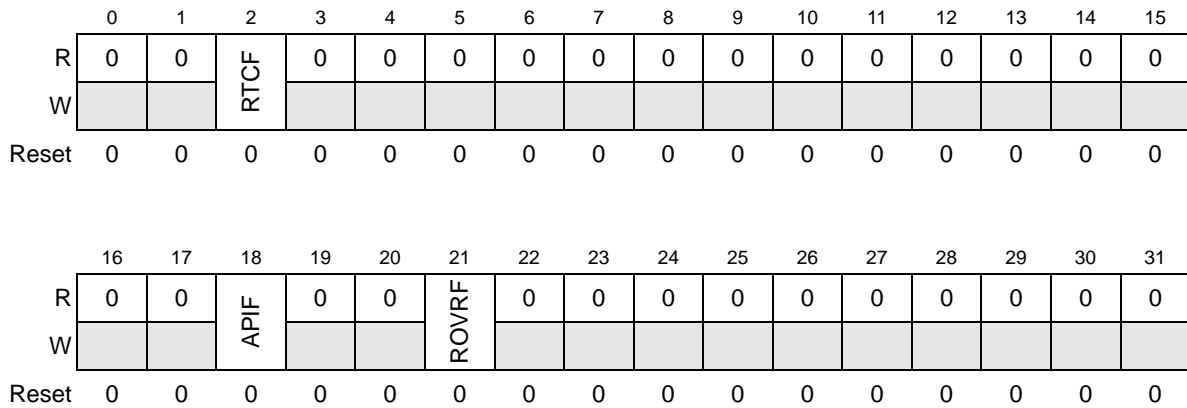


Figure 112. RTC Status Register (RTCS)

Table 110. RTCS field descriptions

Field	Description
RTCF	<p>RTC Interrupt Flag</p> <p>The RTCF bit indicates that the RTC counter has reached the counter value matching RTCVAL. RTCF is cleared by writing a 1 to RTCF. Writing a 0 to RTCF has no effect.</p> <p>1 RTC counter matches RTCVAL 0 RTC counter is not equal to RTCVAL</p>
APIF	<p>API Interrupt Flag</p> <p>The APIF bit indicates that the RTC counter has reached the counter value matching API offset value. APIF is cleared by writing a 1 to APIF. Writing a 0 to APIF has no effect.</p> <p>1 API interrupt 0 No API interrupt</p> <p>Note: The periodic interrupt comes after APIVAL[0:9] + 1'b1 RTC counts</p>
ROVRF	<p>Counter Roll Over Interrupt Flag</p> <p>The ROVRF bit indicates that the RTC has rolled over from 0xffff_ffff to 0x0000_0000. ROVRF is cleared by writing a 1 to ROVRF.</p> <p>1 RTC has rolled over 0 RTC has not rolled over</p>

### 13.5.4 RTC Counter Register (RTCCNT)

The RTCCNT register contains the current value of the RTC counter.

Offset: 0xC

Access: Read

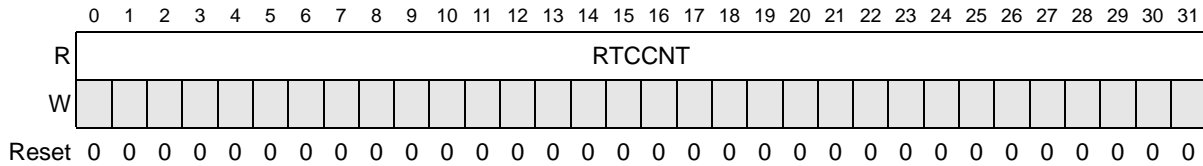


Figure 113. RTC Counter Register (RTCCNT)

Table 111. RTCCNTfield descriptions

Field	Description
RTCCNT	RTC Counter Value Due to the clock synchronization, the RTCCNT value may actually represent a previous counter value.

### 13.6 RTC functional description

The RTC consists of a 32-bit free running counter enabled with the RTCC[**CNTEN**] bit (CNTEN when negated asynchronously resets the counter and synchronously enables the counter when enabled). The value of the counter may be read via the RTCCNT register. Note that due to the clock synchronization, the RTCCNT value may actually represent a previous counter value. The difference between the counter and the read value depends on ratio of counter clock and system clock. Maximum possible difference between the two is 6 count values.

The clock source to the counter is selected with the RTCC[**CLKSEL**] field, which gives the options for clocking the RTC/API. The output of the clock mux can be optionally divided by combination of 512 and 32 to give a 1 ms RTC/API count period for different clock sources. Note that the RTCC[**CNTEN**] bit must be disabled when the RTC/API clock source is switched.

When the counter value for counter bits 10:21 match the 12-bit value in the RTCC[**RTCVAL**] field, then the RTCS[**RTCF**] interrupt flag bit is set (after proper clock synchronization). If the RTCC[**RTCIE**] interrupt enable bit is set, then the RTC interrupt request is generated. The RTC supports interrupt requests in the range of 1 s to 4096 s (> 1 hr) with a 1 s resolution. If there is a match while in low power mode then the RTC will first generate a wakeup request to force a wakeup to run mode, then the RTCF flag will be set.

A rollover wakeup and/or interrupt can be generated when the RTC transitions from a count of 0xFFFF\_FFFF to 0x0000\_0000. The rollover flag is enabled by setting the RTCC[**ROVREN**] bit. An RTC counter rollover with this bit will cause a wakeup from low power mode. An interrupt request is generated for an RTC counter rollover when both the RTCC[**ROVREN**] and RTCC[**RTCIE**] bits are set.

All the flags and counter values are synchronized with the system clock. It is assumed that the system clock frequency is always more than or equal to the rtc\_clk used to run the counter.

# 14 CAN Sampler

## 14.1 Introduction

The CAN Sampler peripheral has been designed to store the first identifier of CAN message “detected” on the CAN bus while no precise clock (Crystal) is running at that time on the device, typically in Low Power modes (STOP, HALT or STANBY) or in RUN mode with crystal switched off.

Depending on both CAN baudrate and Low Power mode used, it is possible to catch either the first or the second CAN frame by sampling one CAN Rx port among 6 and storing all samples in internal registers.

After selection of the mode (first or second frame), the CAN Sampler stores samples of the 48 bits or skips the first frame and stores samples of the 48 bits of second frame using the 16 MHz fast internal RC oscillator and the 5-bit clock prescaler.

After completion, software must process the sampled data in order to rebuild the 48 minimal bits.

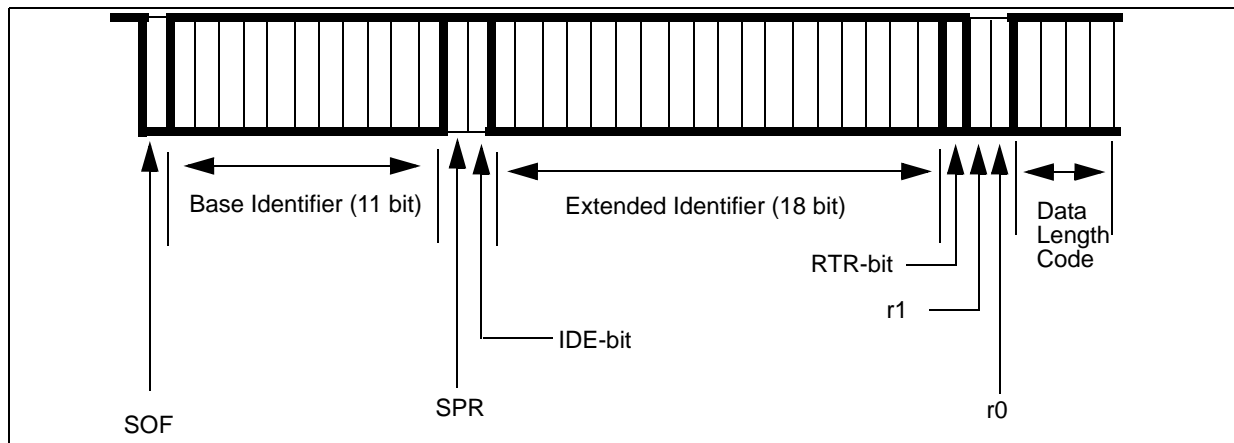


Figure 114. Extended CAN data frame

## 14.2 Main features

- Store 384 samples, equivalent to 48 CAN bit @ 8 samples/bit
- Sample frequency from 500 kHz up to 16 MHz, equivalent at 8 samples/bit to CAN baudrates of 62.5 Kbps to 2 Mbps
- User selectable CAN Rx sample port [CAN0RX-CAN5RX]
- 16 MHz fast internal RC oscillator clock
- 5-bit clock prescaler
- Configurable trigger mode (immediate, next frame)
- Flexible samples processing by software
- Very low power consumption

### 14.3 Memory map and register description

The CAN Sampler registers are listed in [Table 112](#).

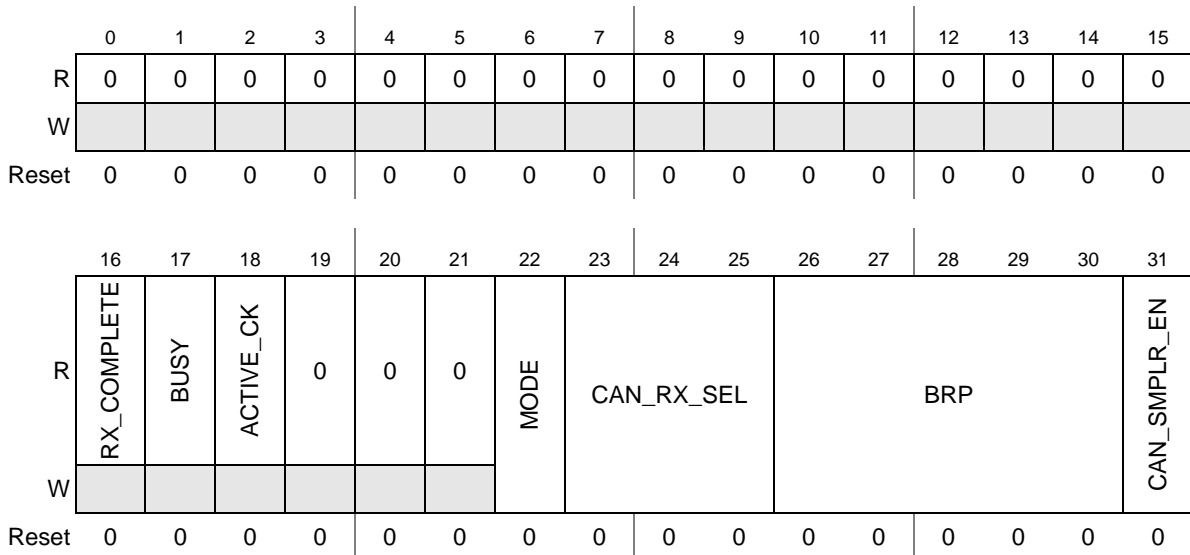
**Table 112. CAN Sampler memory map**

Base address: 0xFFE7_0000		
Address offset	Register	Location
0x00	Control Register (CR)	<a href="#">on page 292</a>
0x04–0x30	Sample registers 0–11	<a href="#">on page 293</a>

#### 14.3.1 Control Register (CR)

Offset: 0x00

Access: Read/write



**Figure 115. Control Register (CR)**

**Table 113. CR field descriptions**

Field	Description
RX_COMPLETE	0: CAN frame has not been stored in the sample registers 1: CAN frame is stored in the sample registers
BUSY	This bit indicates the status of sampling 0: Sampling is complete or has not started 1: Sampling is ongoing
ACTIVE_CK	This bit indicates which is current clock for sample registers i.e xmem_ck. 0: ipg_clk_s is currently xmem_ck 1: RC_CLK is currently xmem_ck
MODE	0: Skip the first frame and sample and store the second frame (SF_MODE) 1: Sample and store the first frame (FF_MODE)

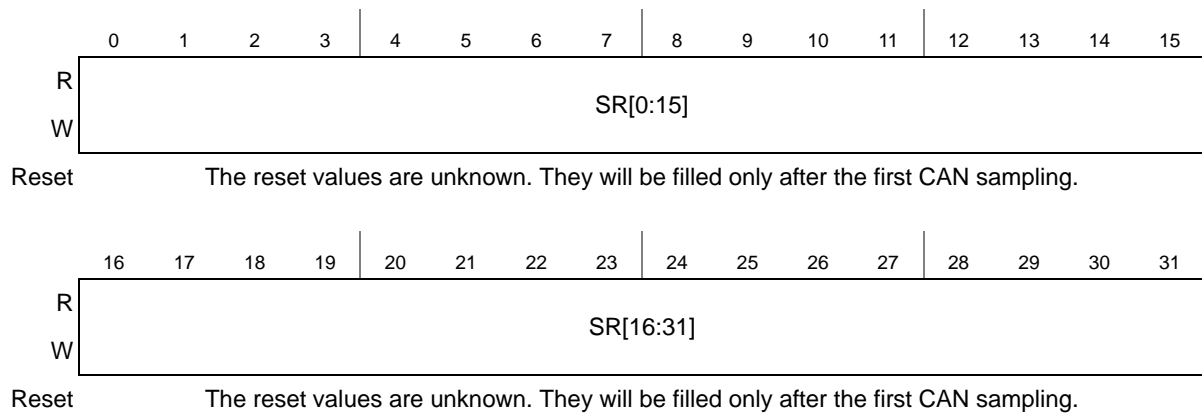
**Table 113. CR field descriptions (continued)**

Field	Description
CAN_RX_SEL	These bits determine which RX bit is sampled. 000: Rx0 is selected 001: Rx1 is selected 010: Rx2 is selected 011: Rx3 is selected 100: Rx4 is selected 101: Rx5 is selected 110: Rx6 is selected 111: Rx7 is selected
BRP	Baudrate Prescaler These bits are used to set the baudrate before going into standby mode 00000: Prescaler has 1 11111: Prescaler has 32
CAN_SMPLR_EN	CAN SAMPLER Enable This bit enables the CAN Sampler before going into standby or stop mode. 0: CAN Sampler is disabled 1: Can Sampler is enabled

**14.3.2 CAN Sampler Sample Registers 0–11**

Offsets: 0x04–0x30 (12 registers)

Access: Read/write



**Figure 116. CAN Sampler Sample Registers 0–11**

## 14.4 Functional description

As the CAN Sampler is driven by the 16 MHz fast internal RC oscillator to sample properly the CAN identifier, two modes are possible depending on both CAN baudrate and Low Power mode used:

- Immediate sampling on falling edge detection (first CAN frame): this mode is used when the fast internal RC oscillator 16 MHz is available in LP mode, e.g. STOP or HALT.
- Sampling on next frame (second CAN frame): this mode is used when the fast internal RC oscillator 16 MHz is switched off in LP mode, e.g. STANDBY. Due to the start-up times of both the Voltage regulator and the fast internal RC oscillator 16 MHz (~10µs), the CAN sampler would miss the first bits of a CAN identifier sent at 500 kbps. Therefore the first identifier is ignored and the sampling is performed on the first falling edge of after interframe space.

The CAN sampler performs sampling on a user selected CAN Rx port among six Rx ports available, normally when the device is in standby or stop mode storing the samples in internal registers. The user is required to configure the baudrate to achieve eight samples per CAN nominal bit. It does not perform any sort of filtering on input samples. Thereafter the software must enable the sampler by setting CAN\_SMPLR\_EN bit in CR register. It then becomes the master controller for accessing the internal registers implemented for storing samples.

The CAN sampler, when enabled, waits for a low pulse on the selected Rx line, taking it as a valid bit of the first CAN frame and generates the RC wakeup request which can be used to start the RC oscillator. Depending upon the mode, it stores the first 8 samples of the 48 bits on selected Rx line or skips the first frame and stores 8 bits for first 48 bits of second frame. In FF\_MODE, it samples the CAN Rx line on RC clock and stores the 8 samples of first 48 bits (384 samples). In SF\_MODE, it samples the Rx and waits for 11 consecutive dominant bits (11 \* 8 samples), taking it as the end of first frame. It then waits for first low pulse on the Rx, taking it as a valid Start of Frame (SOF) of the second frame. The sampler takes 384 samples (48 bytes \* 8) using the RC clock (configuring 8 samples per nominal bit) of the second frame, including the SOF bit. These samples are stored in consecutive addresses of the (12 x 32) internal registers. RX\_COMPLETE bit is set to '1', indicating that sampling is complete.

Software should now process the sampled data by first becoming master for accessing samples internal registers by resetting CAN\_SMPLR\_EN bit. The sampler will need to be enabled again to start waiting for a new sampling routine.

### 14.4.1 Enabling/disabling the CAN sampler

The CAN sampler is disabled on reset and the CPU is able to access the 12 registers used for storing samples. The CAN Sampler must be enabled before going into standby or stop mode by setting CAN\_SMPLR\_EN bit in the Control Register (CR) by writing '1' to this bit.

Any activity on selected Rx line, the sampler enables the 16 MHz fast internal RC oscillator. When CAN\_SMPLR\_EN is reset to 0, the sampler should at least receive three RC clock pulses to reset itself, after which the RC can be switched off.

When the software wishes to access the sample registers contents it must first reset the CAN\_SMPLR\_EN bit by writing a '0'. Before accessing the register contents it must monitor Active\_CK bit for '0'. When this bit is reset it can safely access the (12 x 32) sample

registers. While shifting from normal to sample mode and vice versa, the sample register signals must be static and inactive to ensure the data is not corrupt.

### 14.4.2 Selecting the Rx port

One Rx port can be selected per sampling routine, which port to be sampled is selected by CAN\_RX\_SEL.

**Table 114. Internal multiplexer correspondence**

CAN_RX_SEL	Rx selected
000	CAN0RX on PB[1]
001	CAN1RX on PC[11]
010	CAN2RX on PE[9]
011	CAN3RX on PE[9]
100	CAN4RX on PC[11]
101	CAN5RX on PE[0]
110	Rx6
111	Rx7

### 14.4.3 Baudrate generation

Sampling is performed at a baudrate that is set by the software as a multiple of RC oscillator frequency of 62.5 ns (assuming RC is configured for high frequency mode, that is, 16 MHz). User must set the baudrate prescaler (BRP) such that 8 samples per bit are achieved.

Baudrate setting must be made by software before going into standby or stop mode. This is done by setting BRP bits 5:1 in Control register. The reset value of BRP is 00000 and can be set to max. 11111 which gives a prescale value of BRP+1 thus providing a BRP range of 1 to 32.

- Max. bitrate supported for sampling is 2 Mbps using BRP as 1
- Min. bitrate supported for sampling is 62.5 kbps using BRP as 32

For example, suppose system is transmitting at 125 kbps. In this case, nominal bit period:

$$\text{Equation 2 } T = 1 / (125 \times 10^3) \text{ s} = 8 \times 10^{-3} \times 10^{-3} \text{ s} = 8 \mu \text{ s}$$

To achieve 8 samples per bit

$$\text{Sample period} = 8 / 8 \mu \text{ s} = 1 \mu \text{ s}$$

$$\text{BRP} = 1 \mu \text{ s} / 62.5 \text{ ns} = 16. \text{ Thus in this case BRP} = 01111$$

## 15 e200z0h Core

### 15.1 Overview

The e200 processor family is a set of CPU cores that implement cost-efficient versions of the Power Architecture®. e200 processors are designed for deeply embedded control applications which require low cost solutions rather than maximum performance.

The e200z0h processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in some cases.

The e200z0h core is a single-issue, 32-bit Power Architecture technology VLE-only design with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs).

Instead of the base Power Architecture technology support, the e200z0h core only implements the VLE (variable-length encoding) APU, providing improved code density.

### 15.2 Microarchitecture summary

The e200z0h processor utilizes a four stage pipeline for instruction execution. The Instruction Fetch (stage 1), Instruction Decode/Register file Read/Effective Address Calculation (stage 2), Execute/Memory Access (stage 3), and Register Writeback (stage 4) stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), an 8x32 Hardware Multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching from the BTB is performed to accelerate certain taken branches in the e200z0h. Prefetched instructions are placed into an instruction buffer with 4 entries in e200z0h, each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches which are not taken execute in a single clock. Branches with successful target prefetching have an effective execution time of one clock on e200z0h. All other taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow



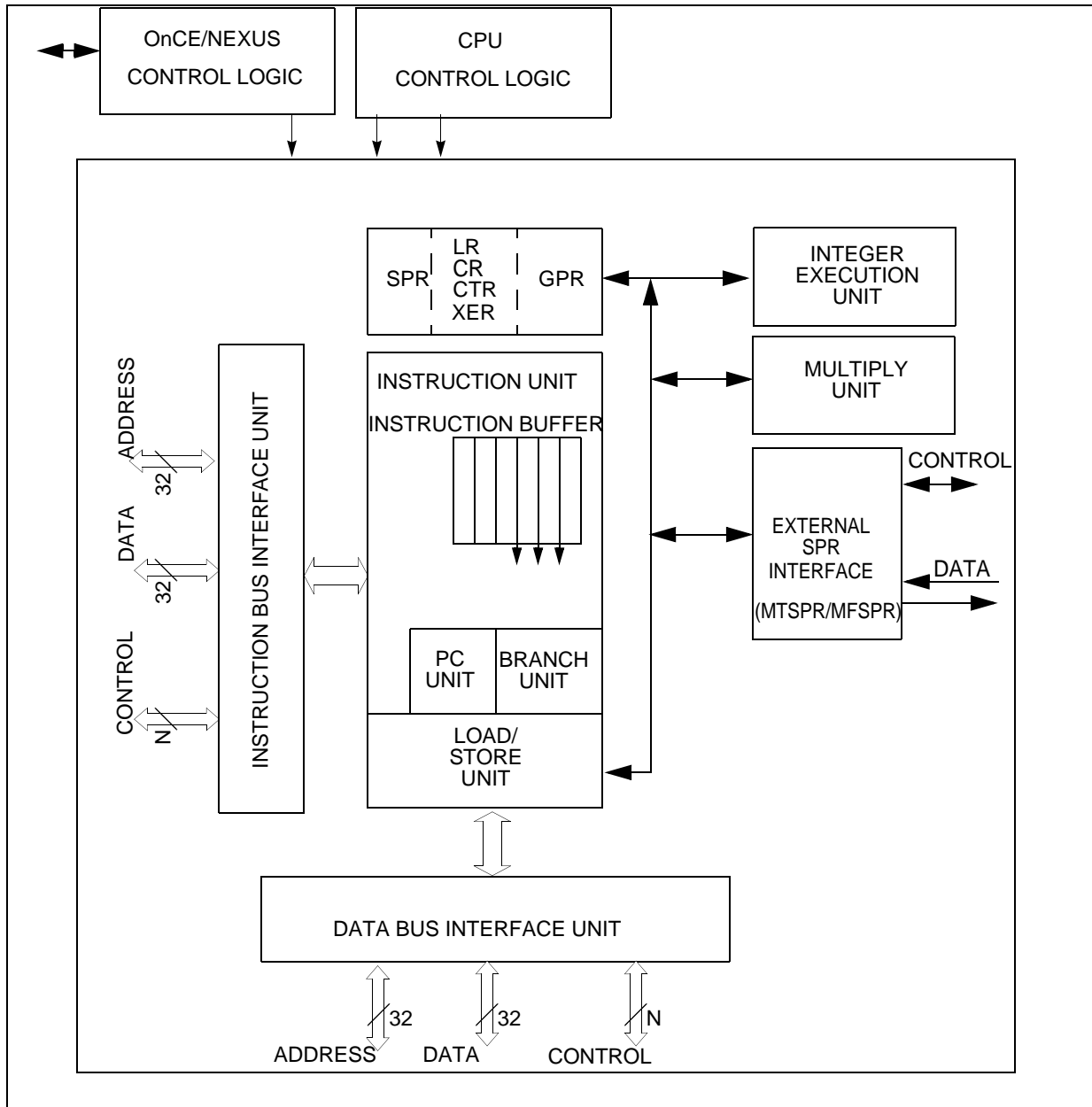
effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture platform. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

### 15.3 Block diagram

Figure 117. e200z0h block diagram



## 15.4 Features

The following is a list of some of the key features of the e200z0h core:

- 32-bit Power Architecture VLE-only programmer's model
- Single issue, 32-bit CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch acceleration using Branch Target Buffer
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and Flash memory via independent Instruction and Data bus interface units (BIUs) (e200z0h only).
- Load/store unit
  - 1 cycle load latency
  - Fully pipelined
  - Big-endian support only
  - Misaligned access support
  - Zero load-to-use pipeline bubbles for aligned transfers
- Power management
  - Low power design
  - Power saving modes: nap, sleep, and wait
  - Dynamic power management of execution units
- Testability
  - Synthesizeable, full MuxD scan design
  - ABIST/MBIST for optional memory arrays

### 15.4.1 Instruction unit features

The features of the e200 Instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or up to two 16-bit VLE instructions per clock
- Instruction buffer with 4 entries in e200z0h, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder supporting single cycle of execution of certain branches, two cycles for all others

### 15.4.2 Integer unit features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 5 to 34 clocks with minimized execution timing
- 8x32 hardware multiplier array supports 1 to 4 cycle 32x32->32 multiply (early out)

### 15.4.3 Load/Store unit features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- 32-bit interface to memory (dedicated memory interface on e200z0h)

### 15.4.4 e200z0h system bus features

The features of the e200z0h system bus interface are as follows:

- Independent instruction and data buses
- AMBA<sup>(m)</sup> AHB<sup>(n)</sup> Lite Rev 2.0 specification with support for ARM v6 AMBA extensions
  - Exclusive access monitor
  - Byte lane strobes
  - Cache allocate support
- 32-bit address bus plus attributes and control on each bus
- 32-bit read data bus for instruction interface
- Separate uni-directional 32-bit read data bus and 32-bit write data bus for data interface
- Overlapped, in-order accesses

### 15.4.5 Nexus 2+ features

The Nexus 2+ module is compliant with Class 2 of the IEEE-ISTO 5001-2003 standard, with additional Class 3 and Class 4 features available. The following features are implemented:

- Program Trace via Branch Trace Messaging (BTM)—Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing

---

m. Advanced Microcontroller Bus Architecture

n. Advanced High Performance Bus

the development tool to interpolate what transpires between the discontinuities. Thus, static code may be traced.

- Ownership Trace via Ownership Trace Messaging (OTM)—OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An Ownership Trace Message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to the processor memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint Messaging via the auxiliary interface
- Watchpoint Trigger enable of Program Trace Messaging
- Auxiliary interface for higher data input/output
  - Configurable (min/max) Message Data Out pins (**nex\_mdo[n:0]**)
  - One (1) or two (2) Message Start/End Out pins (**nex\_mseo\_b[1:0]**)
  - One (1) Read/Write Ready pin (**nex\_rdy\_b**) pin
  - One (1) Watchpoint Event pin (**nex\_evto\_b**)
  - One (1) Event In pin (**nex\_evti\_b**)
  - One (1) MCKO (Message Clock Out) pin
- Registers for Program Trace, Ownership Trace and Watchpoint Trigger control
- All features controllable and configurable via the JTAG port

## 15.5 Core registers and programmer's model

This section describes the registers implemented in the e200z0h cores. It includes an overview of registers defined by the Power Architecture platform, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in the Power Architecture specification.

The Power Architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

*Figure 118*, and *Figure 117* show the e200 register set including the registers which are accessible while in supervisor mode, and the registers which are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

*Note:* *e200z0h is a 32-bit implementation of the Power Architecture specification. In this document, register bits are sometimes numbered from bit 0 (Most Significant Bit) to 31 (Least Significant Bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher.*

*Where appropriate, the Book E defined bit numbers are shown in parentheses.*

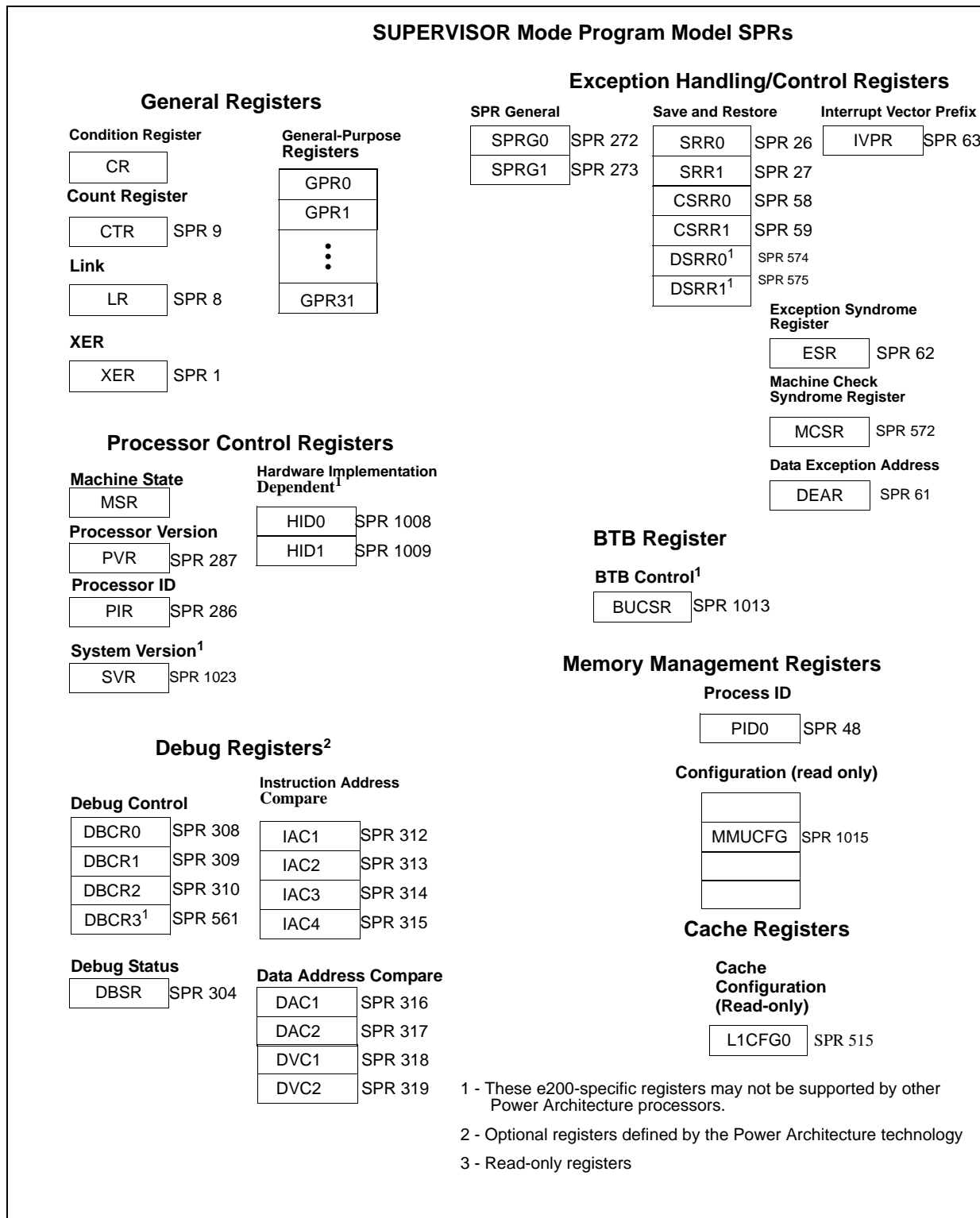


Figure 118. e200z0 SUPERVISOR Mode Program Model SPRs

## 16 Enhanced Direct Memory Access (eDMA)

### 16.1 Device-specific features

- 16 programmable channels to support independent 8, 16 or 32-bit single value or block transfers
- Support of variable sized queues and circular queues
- Source and destination address registers independently configured to post-incrementor remain constant
- Each transfer initiated by peripheral, CPU, periodic timer interrupt or eDMA channel request
- Peripheral eDMA request sources possible from DSPI, I<sup>2</sup>C, 10-bit ADC, 12-bit ADC, LINFlexD, and eMIOS
- Each eDMA channel able to optionally send interrupt request to CPU on completion of single value or block transfer
- DMA transfers possible between system memories and all accessible memory mapped locations including peripheral and registers
- Programmable eDMA Channel Mux allows assignment of any eDMA source to any available eDMA channel with total of up to 32 request sources
- DMA supports the following functionality:
  - Scatter Gather
  - Channel Linking
  - Inner Loop Offset
  - Arbitration
    - Fixed Group, fixed channel
    - Round Robin Group, fixed channel
    - Round Robin Group, Round Robin Channel
    - Fixed Group, Round Robin Channel
  - Channel preemption
  - Cancel channel transfer
- Interrupts – The eDMA has a single interrupt request for each implemented channel and a combined eDMA Error interrupt to flag transfer errors to the system. Each channel eDMA interrupt can be enabled or disabled and provides notification of a completed transfer. Refer to the Interrupt Vector in the Interrupt Controller chapter of the reference manual for the allocation of these interrupts.

### 16.2 Introduction

The enhanced direct memory access controller (eDMA) is a second-generation platform block capable of performing complex data movements through 16 programmable channels, with minimal intervention from the host processor. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation minimizes the overall block size.

Figure 119 is a block diagram of the eDMA module.

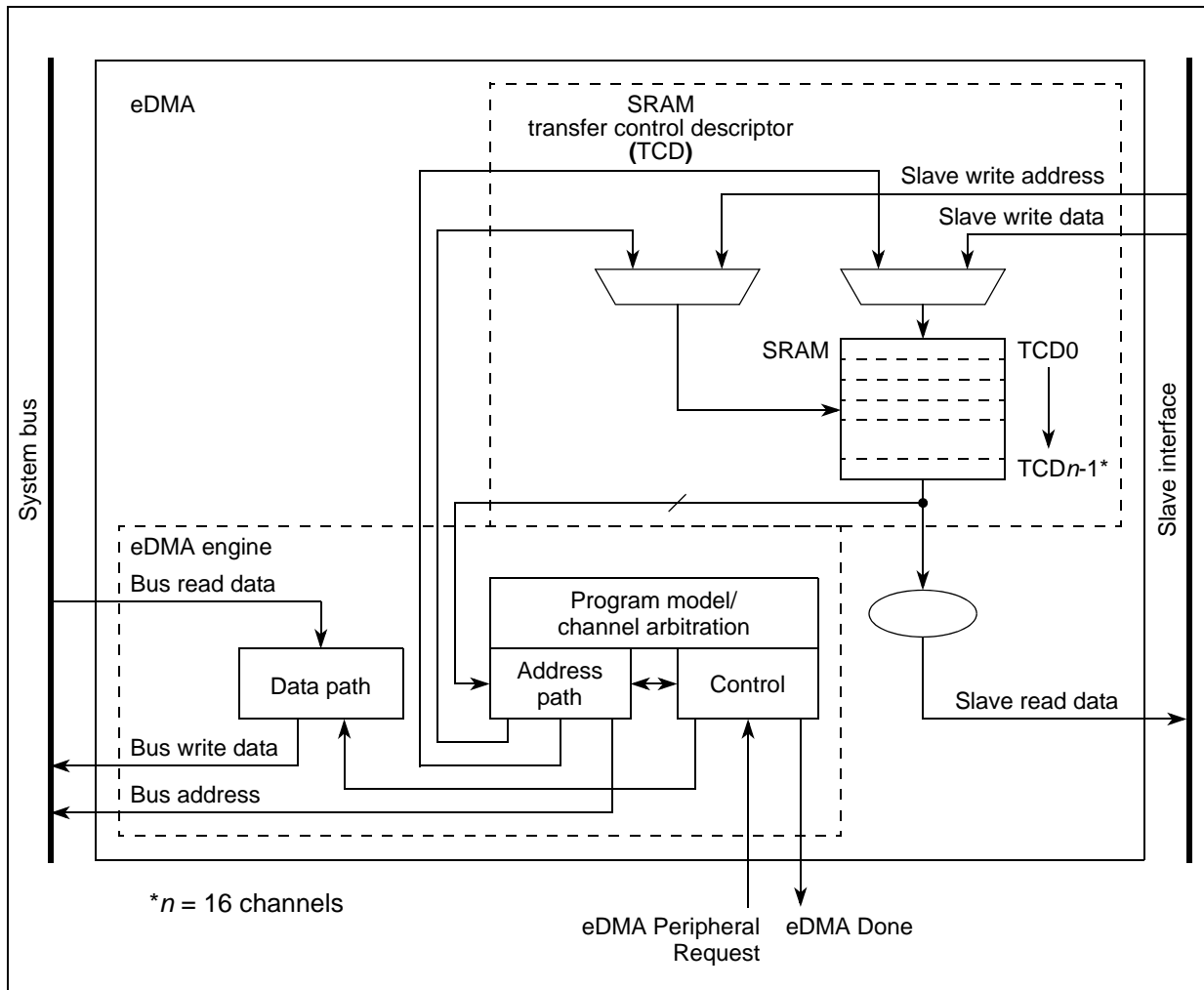


Figure 119. eDMA block diagram



## 16.2.1 Features

The eDMA module supports the following features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An *inner* data transfer loop defined by a “minor” byte transfer count
  - An *outer* data transfer loop defined by a “major” iteration count
- Channel service request via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
    - Independent channel linking at end of minor loop and/or major loop
  - Peripheral-paced hardware requests (one per channel)
  - For all three methods, *one service request per execution of the minor loop is required*
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel, and logically summed together to form a small number of error interrupt outputs
- Support for scatter/gather eDMA processing
- Support for complex data structures
- Support to cancel transfers via software or hardware

## 16.3 Memory map and register definition

### 16.3.1 Memory map

The eDMA memory map is shown in [Table 115](#). The eDMA base address is 0xFFF4\_4000. The address of each register is given as an offset to the eDMA base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

The eDMA's programming model is partitioned into two regions—the first region defines a number of registers providing control functions; the second region corresponds to the local transfer control descriptor memory.

**Table 115. eDMA memory map**

Base address: 0xFFF4_4000		
Address offset	Register	Location
0x0000	EDMA_CR — eDMA control register	<a href="#">on page 307</a>
0x0004	EDMA_ESR — eDMA error status register	<a href="#">on page 309</a>
0x0008	Reserved	

**Table 115. eDMA memory map (continued)**

Base address: 0xFFFF4_4000		
Address offset	Register	Location
0x000C	EDMA_ERQRL — eDMA enable request low register (channels 15–00)	<a href="#">on page 311</a>
0x0010	Reserved	
0x0014	EDMA_EEIRL — eDMA enable error interrupt low register (channels 15–00)	<a href="#">on page 312</a>
0x0018	EDMA_SERQR — eDMA set enable request register	<a href="#">on page 313</a>
0x0019	EDMA_CERQR — eDMA clear enable request register	<a href="#">on page 314</a>
0x001A	EDMA_SEEIR — eDMA set enable error interrupt register	<a href="#">on page 314</a>
0x001B	EDMA_CEEIR — eDMA clear enable error interrupt register	<a href="#">on page 315</a>
0x001C	EDMA_CIRQR — eDMA clear interrupt request register	<a href="#">on page 315</a>
0x001D	EDMA_CER — eDMA clear error register	<a href="#">on page 316</a>
0x001E	EDMA_SBR — eDMA set start bit register	<a href="#">on page 316</a>
0x001F	EDMA_CDSBR — eDMA clear done status bit register	<a href="#">on page 317</a>
0x0020	Reserved	
0x0024	EDMA_IRQRL — eDMA interrupt request low register	<a href="#">on page 317</a>
0x0028	Reserved	
0x002C	EDMA_ERL — eDMA error low register	<a href="#">on page 318</a>
0x0030	Reserved	
0x0034	EDMA_HRSL — eDMA hardware request status register	<a href="#">on page 319</a>
0x0038 – 0x01FC	Reserved	
0x0100	EDMA_CPR0 — eDMA channel 0 priority register	<a href="#">on page 319</a>
0x0101	EDMA_CPR1 — eDMA channel 1 priority register	<a href="#">on page 319</a>
0x0102	EDMA_CPR2 — eDMA channel 2 priority register	<a href="#">on page 319</a>
0x0103	EDMA_CPR3 — eDMA channel 3 priority register	<a href="#">on page 319</a>
0x0104	EDMA_CPR4 — eDMA channel 4 priority register	<a href="#">on page 319</a>
0x0105	EDMA_CPR5 — eDMA channel 5 priority register	<a href="#">on page 319</a>
0x0106	EDMA_CPR6 — eDMA channel 6 priority register	<a href="#">on page 319</a>
0x0107	EDMA_CPR7 — eDMA channel 7 priority register	<a href="#">on page 319</a>
0x0108	EDMA_CPR8 — eDMA channel 8 priority register	<a href="#">on page 319</a>
0x0109	EDMA_CPR9 — eDMA channel 9 priority register	<a href="#">on page 319</a>
0x010A	EDMA_CPR10 — eDMA channel 10 priority register	<a href="#">on page 319</a>
0x010B	EDMA_CPR11 — eDMA channel 11 priority register	<a href="#">on page 319</a>
0x010C	EDMA_CPR12 — eDMA channel 12 priority register	<a href="#">on page 319</a>
0x010D	EDMA_CPR13 — eDMA channel 13 priority register	<a href="#">on page 319</a>
0x010E	EDMA_CPR14 — eDMA channel 14 priority register	<a href="#">on page 319</a>



Table 115. eDMA memory map (continued)

Base address: 0xFFFF4_4000		
Address offset	Register	Location
0x010F	EDMA_CPR15 — eDMA channel 15 priority register	<a href="#">on page 319</a>
0x0110	Reserved	
0x1000	TCD00 — eDMA transfer control descriptor 00	<a href="#">on page 321</a>
0x1020	TCD01 — eDMA transfer control descriptor 01	<a href="#">on page 321</a>
0x1040	TCD02 — eDMA transfer control descriptor 02	<a href="#">on page 321</a>
0x1060	TCD03 — eDMA transfer control descriptor 03	<a href="#">on page 321</a>
0x1080	TCD04 — eDMA transfer control descriptor 04	<a href="#">on page 321</a>
0x10A0	TCD05 — eDMA transfer control descriptor 05	<a href="#">on page 321</a>
0x10C0	TCD06 — eDMA transfer control descriptor 06	<a href="#">on page 321</a>
0x10E0	TCD07 — eDMA transfer control descriptor 07	<a href="#">on page 321</a>
0x1100	TCD08 — eDMA transfer control descriptor 08	<a href="#">on page 321</a>
0x1120	TCD09 — eDMA transfer control descriptor 09	<a href="#">on page 321</a>
0x1140	TCD10 — eDMA transfer control descriptor 10	<a href="#">on page 321</a>
0x1160	TCD11 — eDMA transfer control descriptor 11	<a href="#">on page 321</a>
0x1180	TCD12 — eDMA transfer control descriptor 12	<a href="#">on page 321</a>
0x11A0	TCD13 — eDMA transfer control descriptor 13	<a href="#">on page 321</a>
0x11C0	TCD14 — eDMA transfer control descriptor 14	<a href="#">on page 321</a>
0x11E0	TCD15 — eDMA transfer control descriptor 15	<a href="#">on page 321</a>
0x1200	Reserved	

## 16.3.2 Register descriptions

### 16.3.2.1 DMA Control Register (EDMA\_CR)

The 32-bit EDMA\_CR defines the basic operating configuration of the eDMA.

Arbitration among the channels can be configured to use a fixed priority or a round robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see [Section 16.3.2.16: DMA Channel n Priority \(EDMA\\_CPRn\)](#)). In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through, from channel 15 down to channel 0, without regard to priority.

See [Figure 120](#) and [Table 116](#) for the EDMA\_CR definition.

Offset: 0x0000

Access: Read/write

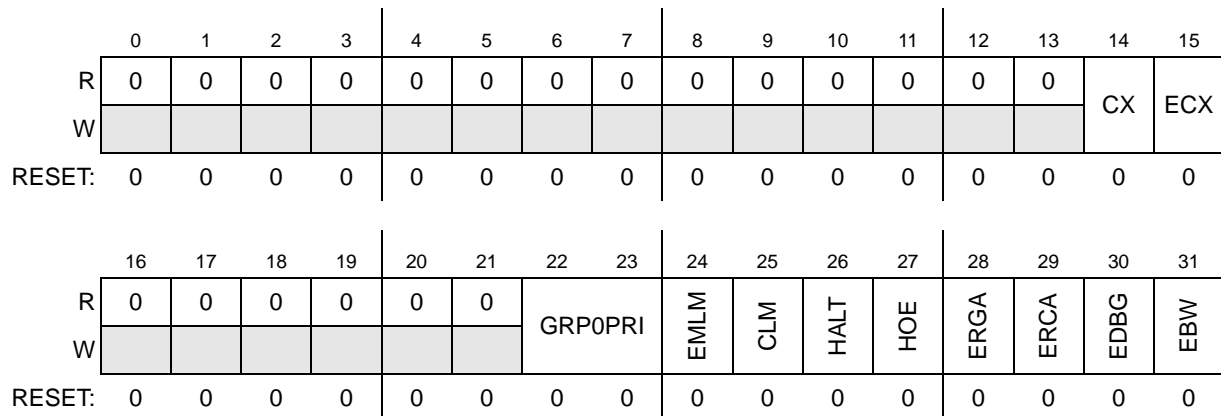


Figure 120. DMA Control Register (EDMA\_CR)

Table 116. EDMA\_CR field descriptions

Field	Description
CX	<p>Cancel Transfer</p> <p>0 Normal operation.</p> <p>1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CXFR bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.</p>
ECX	<p>Error Cancel Transfer</p> <p>0 Normal operation.</p> <p>1 Cancel the remaining data transfer in the same fashion as the CX cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the EDMA_ESR register and generating an optional error interrupt (see <a href="#">Section 16.3.2.2: DMA Error Status (EDMA_ESR)</a>).</p>
GRP0PRI	<p>Channel Group 0 Priority</p> <p>Group 0 priority level when fixed priority group arbitration is enabled.</p>
EMLM	<p>Enable Minor Loop Mapping</p> <p>0 Minor loop mapping disabled. TCDn.word2 is defined as a 32-bit nbytes field.</p> <p>1 Minor loop mapping enabled. When set, TCDn.word2 is redefined to include individual enable fields, an offset field and the nbytes field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The nbytes field is reduced when either offset is enabled.</p>
CLM	<p>Continuous Link Mode</p> <p>0 A minor loop channel link made to itself will go through channel arbitration before being activated again.</p> <p>1 A minor loop channel link made to itself will not go through channel arbitration before being activated again. Upon minor loop completion the channel will active again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.</p>

Table 116. EDMA\_CR field descriptions (continued)

Field	Description
HALT	Halt DMA Operations 0 Normal operation. 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution will resume when the HALT bit is cleared.
HOE	Halt On Error 0 Normal operation. 1 Any error will cause the HALT bit to be set. Subsequently, all service requests will be ignored until the HALT bit is cleared.
ERGA	Enable Round Robin Group Arbitration 0 Fixed priority arbitration is used for selection among the groups. 1 Round robin arbitration is used for selection among the groups.
ERCA	Enable Round Robin Channel Arbitration 0 Fixed priority arbitration is used for channel selection within each group. 1 Round robin arbitration is used for channel selection within each group.
EDBG	Enable Debug 0 The assertion of the device debug mode is ignored. 1 The assertion of the device debug mode causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution will resume when either the device comes out of debug mode or the EDBG bit is cleared.
EBW	0 The bufferable write signal (hprot[2]) is not asserted during AMBA AHB writes. 1 The bufferable write signal (hprot[2]) is asserted on all AMBA AHB writes except for the last write sequence.

### 16.3.2.2 DMA Error Status (EDMA\_ESR)

The EDMA\_ESR provides information about the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer\_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed-arbitration mode, a configuration error is generated when any two channel priority levels are equal and any channel is activated. The ERRCHN field is undefined for this type of error. All channel priority levels must be unique before any service requests are made.

If a scatter-gather operation is enabled on channel completion, a configuration error is reported if the scatter-gather address (DLAST\_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled on channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E\_LINK bit is not equal to the TCD.BITER.E\_LINK bit. All configuration error conditions except scatter-gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter-gather configuration error is reported when the

scatter-gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write will execute using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence will execute before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the DMA engine to stop the active channel and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA\_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the DMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.

Offset: 0x0004

Access: Read

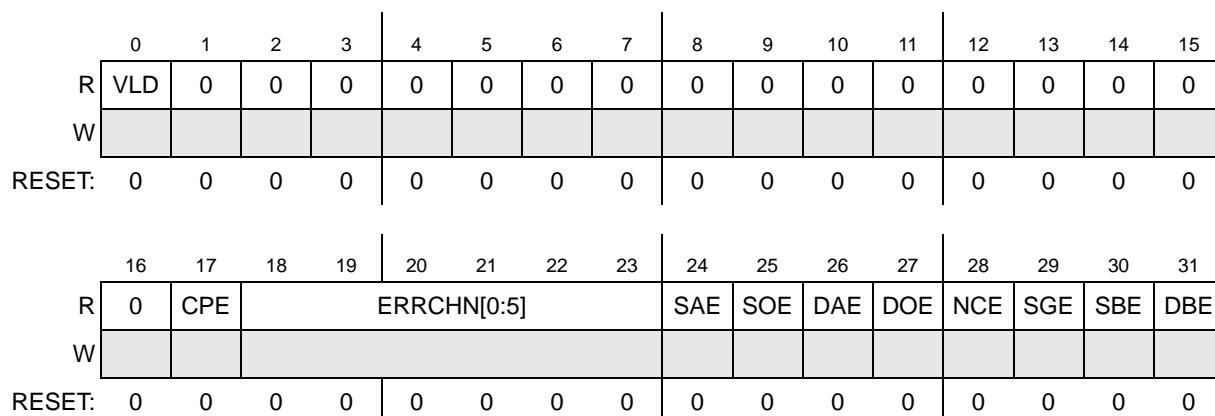


Figure 121. DMA Error Status (EDMA\_ESR) Register

Table 117. EDMA\_ESR field descriptions

Field	Description
VLD	Logical OR of all EDMA_ERL status bits. 0 No EDMA_ERL bits are set. 1 At least one EDMA_ERL bit is set indicating a valid error exists that has not been cleared.
CPE	Channel Priority Error 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group. All channel priorities within a group are not unique.
ERRCHN[0:5]	Error Channel Number or Cancelled Channel Number The channel number of the last recorded error (excluding GPE and CPE errors) or last recorded transfer that was error cancelled.

**Table 117. EDMA\_ESR field descriptions (continued)**

Field	Description
SAE	Source Address Error 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.saddr field. TCD.saddr is inconsistent with TCD.ssize.
SOE	Source Offset Error 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.soff field. TCD.soff is inconsistent with TCD.ssize.
DAE	Destination Address Error 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.daddr field. TCD.daddr is inconsistent with TCD.dsize.
DOE	Destination Offset Error 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.doff field. TCD.doff is inconsistent with TCD.dsize.
NCE	Nbytes/Citer Configuration Error 0 No nbytes/citer configuration error. 1 The last recorded error was a configuration error detected in the TCD.nbytes or TCD.citer fields. TCD.nbytes is not a multiple of TCD.ssize and TCD.dsize, or TCD.citer is equal to zero, or TCD.citer.e_link is not equal to TCD.biter.e_link.
SGE	Scatter/Gather Configuration Error 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.dlast_sga field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.e_sg is enabled. TCD.dlast_sga is not on a 32 byte boundary.
SBE	Source Bus Error 0 No source bus error. 1 The last recorded error was a bus error on a source read.
DBE	Destination Bus Error 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

**16.3.2.3 DMA Enable Request (EDMA\_ERQRL)**

The EDMA\_ERQRL provides a bit map for the 16 channels to enable the request signal for each channel. EDMA\_ERQRL maps to channels 15–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA\_SERQR, and EDMA\_CERQR registers. The EDMA\_CERQR and EDMA\_SERQR registers are provided so the request enable for a single channel can be modified without performing a read-modify-write sequence to the EDMA\_ERQRL register.

Both the eDMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not affect a channel service request made through software or a linked channel request.

Offset: 0x000C

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERQ15	ERQ14	ERQ13	ERQ12	ERQ11	ERQ10	ERQ09	ERQ08	ERQ07	ERQ06	ERQ05	ERQ04	ERQ03	ERQ02	ERQ01	ERQ00
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 122. DMA Enable Request (EDMA\_ERQL) Registers

Table 118. EDMA\_ERQL field descriptions

Field	Description
ERQn	Enable eDMA Request n 0 The eDMA request signal for channel n is disabled. 1 The eDMA request signal for channel n is enabled.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA\_ERQL bit for that channel. If the TCD.d\_req bit is set, then the corresponding EDMA\_ERQL bit is cleared, disabling the eDMA request; else if the d\_req bit is cleared, the state of the EDMA\_ERQL bit is unaffected.

**16.3.2.4 DMA Enable Error Interrupt (EDMA\_EEIRL)**

The EDMA\_EEIRL provides a bit map for the 16 channels to enable the error interrupt signal for each channel. EDMA\_EEIRL maps to channels 15–0.

The state of any given channel’s error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA\_SEEIR and EDMA\_CEEIR registers. The EDMA\_SEEIR and EDMA\_CEEIR registers are provided so that the error interrupt enable for a single channel can be modified without the performing a read-modify-write sequence to the EDMA\_EEIRL register.

Both the eDMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted. See [Figure 123](#) and [Table 119](#) for the EDMA\_EEIRL definition.





Offset: 0x0014

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 123. DMA Enable Error Interrupt (EDMA\_EEIRL) Register

Table 119. EDMA\_EEIRL field descriptions

Field	Description
EEIn	Enable Error Interrupt n 0 The error signal for channel n does not generate an error interrupt. 1 The assertion of the error signal for channel n generate an error interrupt request.

16.3.2.5 DMA Set Enable Request (EDMA\_SERQR)

The EDMA\_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA\_ERQRL to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_ERQRL to be set. Setting bit 1 (SERQ[0]) provides a global set function, forcing the entire contents of EDMA\_ERQRL to be asserted. Reads of this register return all zeroes.

Offset: 0x0018

Access: Write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SERQ						
RESET:	0	0	0	0	0	0	0	0

Figure 124. DMA Set Enable Request (EDMA\_SERQR) Register

Table 120. EDMA\_SERQR field descriptions

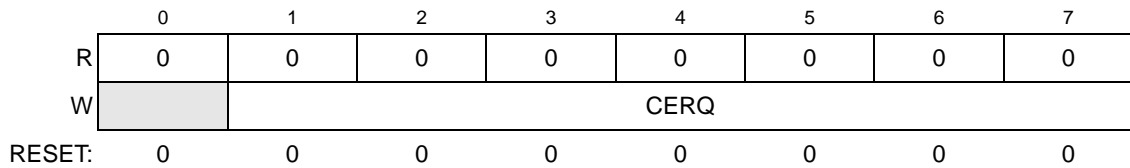
Field	Description
SERQ	Set Enable Request 0-15 Set the corresponding bit in EDMA_ERQRL 64-127 Set all bits in EDMA_ERQRL

**16.3.2.6 DMA Clear Enable Request (EDMA\_CERQR)**

The EDMA\_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA\_ERQRL to disable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_ERQRL to be cleared. Setting bit 1 (CERQ[0]) provides a global clear function, forcing the entire contents of the EDMA\_ERQRL to be zeroed, disabling all eDMA request inputs. Reads of this register return all zeroes. See [Figure 125](#) and [Table 121](#) for the EDMA\_CERQR definition.

Offset: 0x0019

Access: Write



**Figure 125. DMA Clear Enable Request (EDMA\_CERQR) Register**

**Table 121. EDMA\_CERQR field descriptions**

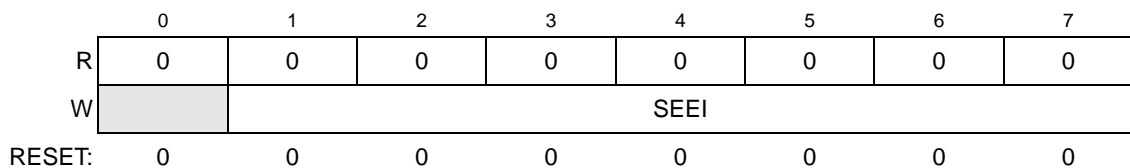
Field	Description
CERQ	Clear Enable Request 0-63 Clear corresponding bit in EDMA_ERQRL 64-127 Clear all bits in EDMA_ERQRL

**16.3.2.7 DMA Set Enable Error Interrupt (EDMA\_SEEIR)**

The EDMA\_SEEIR provides a memory-mapped mechanism to set a given bit in the EDMA\_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_EEIRL to be set. Setting bit 1 (SEEI[0]) provides a global set function, forcing the entire contents of EDMA\_EEIRL to be asserted. Reads of this register return all zeroes. See [Figure 126](#) and [Table 122](#) for the EDMA\_SEEIR definition.

Offset: 0x001A

Access: Write



**Figure 126. DMA Set Enable Error Interrupt (EDMA\_SEEIR) Register**

**Table 122. EDMA\_SEEIR field descriptions**

Name	Description
SEEI	Set Enable Error Interrupt 0-63 Set the corresponding bit in EDMA_EEIRL 64-127 Set all bits in EDMA_EEIRL

### 16.3.2.8 DMA Clear Enable Error Interrupt (EDMA\_CEEIR)

The EDMA\_CEEIR provides a memory-mapped mechanism to clear a given bit in the EDMA\_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_EEIRL to be cleared. Setting bit 1 (CEEI[0]) provides a global clear function, forcing the entire contents of the EDMA\_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register returns all zeroes. See [Figure 127](#) and [Table 123](#) for the EDMA\_CEEIR definition.

Offset: 0x001B

Access: Write

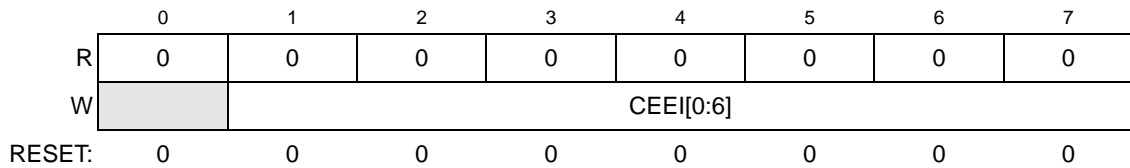


Figure 127. DMA Clear Enable Error Interrupt (EDMA\_CEEIR) Register

Table 123. EDMA\_CEEIR field descriptions

Field	Description
CEEI	Clear Enable Error Interrupt 0-63 Clear corresponding bit in EDMA_EEIRL 64-127 Clear all bits in EDMA_EEIRL

### 16.3.2.9 DMA Clear Interrupt Request (EDMA\_CIRQR)

The EDMA\_CIRQR provides a memory-mapped mechanism to clear a given bit in the EDMA\_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA\_IRQRL to be cleared. Setting bit 1 (CINT[0]) provides a global clear function, forcing the entire contents of the EDMA\_IRQRL to be zeroed, disabling all eDMA interrupt requests. Reads of this register return all zeroes. See [Figure 128](#) and [Table 124](#) for the EDMA\_CIRQR definition.

Offset: 0x001C

Access: Write

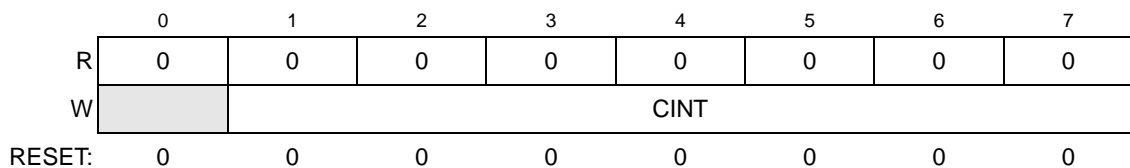


Figure 128. DMA Clear Interrupt Request (EDMA\_CIRQR) Fields

Table 124. EDMA\_CIRQR field descriptions

Field	Description
CINT[0:6]	Clear Interrupt Request 0-63 Clear the corresponding bit in EDMA_IRQRL 64-127 Clear all bits in EDMA_IRQRL

### 16.3.2.10 DMA Clear Error (EDMA\_CER)

The EDMA\_CER provides a memory-mapped mechanism to clear a given bit in the EDMA\_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA\_ERL to be cleared. Setting bit 1 (CERR[0]) provides a global clear function, forcing the entire contents of the EDMA\_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes. See [Figure 129](#) and [Table 125](#) for the EDMA\_CER definition.

Offset: 0x001D

Access: Write

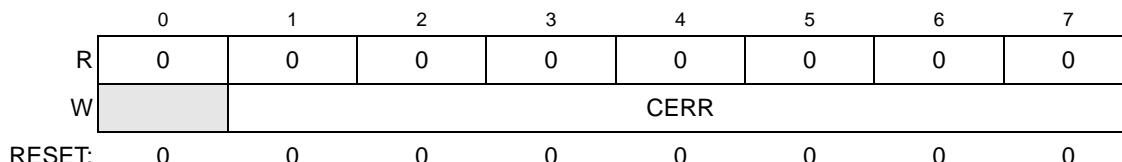


Figure 129. DMA Clear Error (EDMA\_CER) Register

Table 125. EDMA\_CER field descriptions

Field	Description
CERR	Clear Error Indicator 0-63 Clear corresponding bit in EDMA_ERL 64-127 Clear all bits in EDMA_ERL

### 16.3.2.11 DMA Set START Bit (EDMA\_SSBR)

The EDMA\_SSBR provides a memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB[0]) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes. See [Table 133](#) for the TCD START bit definition.

Offset: 0x001E

Access: Write

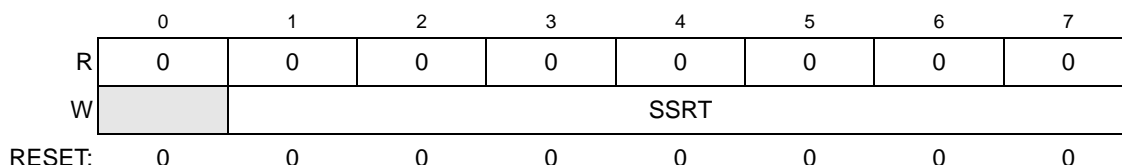


Figure 130. DMA Set START Bit (EDMA\_SSBR) Register

Table 126. EDMA\_SSBR field descriptions

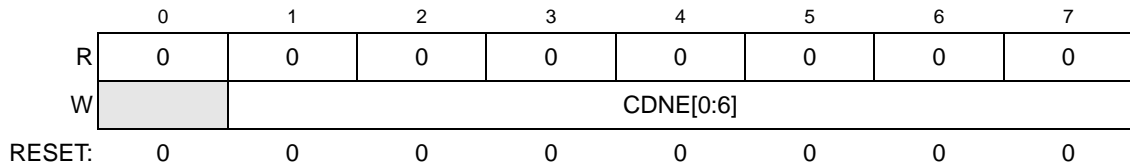
Field	Description
SSRT	Set START Bit (Channel Service Request) 0-63 Set the corresponding channel's TCD.start 64-127 Set all TCD.start bits

**16.3.2.12 DMA Clear DONE Status (EDMA\_CDSBR)**

The EDMA\_CDSBR provides a memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB[0]) provides a global clear function, forcing all DONE bits to be cleared. See [Table 133](#) for the TCD DONE bit definition.

Offset: 0x001F

Access: Write



**Figure 131. DMA Clear DONE Status (EDMA\_CDSBR) Register**

**Table 127. EDMA\_CDSBR field descriptions**

Field	Description
CDNE[0:6]	Clear DONE Status Bit 0-63 Clear the corresponding channel's DONE bit 64-127 Clear all TCD DONE bits

**16.3.2.13 DMA Interrupt Request (EDMA\_IRQRL)**

The EDMA\_IRQRL provides a bit map for the 16 channels signaling the presence of an interrupt request for each channel. EDMA\_IRQRL maps to channels 15–0.

The DMA engine signals the occurrence of a programmed interrupt on the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA\_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA\_CIRQR. On writes to the EDMA\_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA\_CIRQR is provided so the interrupt request for a single channel can be cleared without performing a read-modify-write sequence to the EDMA\_IRQRL. See [Figure 132](#) and [Table 128](#) for the EDMA\_IRQRL definition.

Offset: 0x0024

Access: Read/write

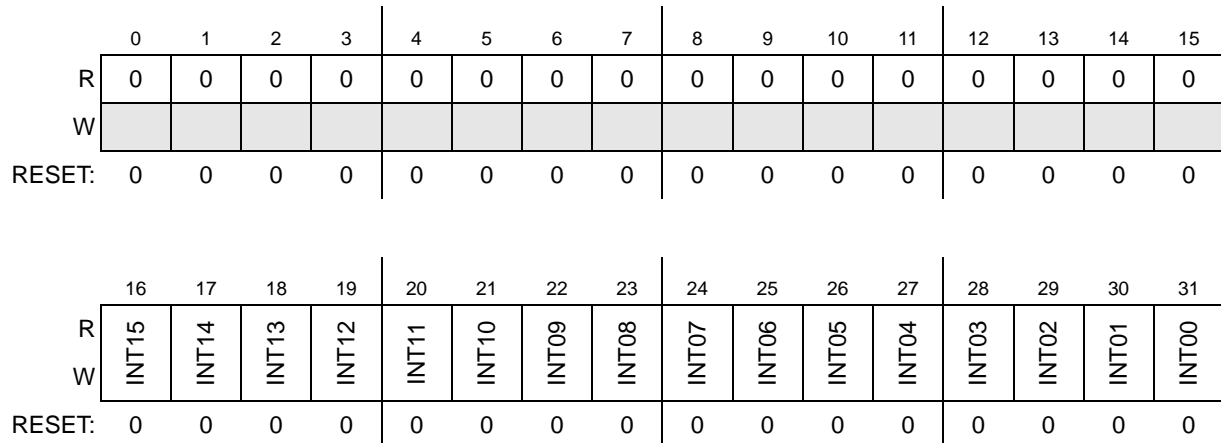


Figure 132. DMA Interrupt Request (EDMA\_IRQRL) Registers

Table 128. EDMA\_IRQRL field descriptions

Field	Description
INTn	DMA Interrupt Request n 0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

16.3.2.14 DMA Error (EDMA\_ERL)

The EDMA\_ERL provides a bit map for the 16 channels signaling the presence of an error for each channel. EDMA\_ERL maps to channels 15-0.

The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA\_EEIR, then logically summed across 16 channels to form an error interrupt request, which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any eDMA errors, it is software’s responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA\_CER in the interrupt service routine is used for this purpose. The normal eDMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA\_EEIR. The EDMA\_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel’s error indicators is affected by writes to this register; it is also affected by writes to the EDMA\_CER. On writes to EDMA\_ERL, a 1 in any bit position clears the corresponding channel’s error status. A 0 in any bit position has no affect on the corresponding channel’s current error status. The EDMA\_CER is provided so the error indicator for a single channel can be cleared. See [Figure 133](#) and [Table 129](#) for the EDMA\_ERL definition.



Offset: 0x002C

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERR15	ERR14	ERR13	ERR12	ERR11	ERR10	ERR09	ERR08	ERR07	ERR06	ERR05	ERR04	ERR03	ERR02	ERR01	ERR00
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 133. DMA Error (EDMA\_ERL) Registers

Table 129. EDMA\_ERL field descriptions

Field	Description
ERRn	DMA Error n 0 An error in channel n has not occurred. 1 An error in channel n has occurred.

16.3.2.15 DMA Hardware Request Status (EDMA\_HRSL)

The EDMA\_HRSL register provides a bit map for the implemented channels to show the current hardware request status for each channel. This view into the hardware request signals may be used for debug purposes.

See [Figure 134](#) and [Table 130](#) for the EDMA\_HRSL definition.

Offset: 0x0034

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HRS15	HRS14	HRS13	HRS12	HRS11	HRS10	HRS09	HRS08	HRS07	HRS06	HRS05	HRS04	HRS03	HRS02	HRS01	HRS00
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 134. DMA Hardware Request Status (EDMA\_HRSL) Register

**Table 130. EDMA\_HRSL field descriptions**

Field	Description
HRSn	DMA Hardware Request Status n 0 A hardware service request for channel n is not present. 1 A hardware service request for channel n is present.  <b>Note:</b> The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[n] bit.

**16.3.2.16 DMA Channel n Priority (EDMA\_CPRn)**

When the fixed-priority channel arbitration mode is enabled (EDMA\_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software modifies channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error will be reported. The range of the priority value is limited to the values of 0 through 15.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA\_CPRn register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel requests service, the restored channel will be suspended and the higher priority channel will be serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected for channel arbitration mode

A channel's ability to preempt another channel can be disabled by setting the DPA bit in the EDMA\_CPRn register. When a channel's preempt ability is disabled, that channel cannot suspend a lower priority channel's data transfer; regardless of the lower priority channel's ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available a true, high priority channel. See [Figure 135](#) and [Table 131](#) for the EDMA\_CPRn definition.

Offset: 0x0100 + n

Access: Read/write



**Figure 135. DMA Channel n Priority (EDMA\_CPRn) Register**



**Table 131. EDMA\_CPRn field descriptions**

Field	Description
ECP	Enable Channel Preemption 0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
DPA	Disable Preempt Ability 0 Channel n can suspend a lower priority channel. 1 Channel n cannot suspend any channel, regardless of channel priority.
CHPRI[0:3]	Channel n Arbitration Priority Channel priority when fixed-priority arbitration is enabled.

**16.3.2.17 Transfer Control Descriptor (TCD)**

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. The definitions of the TCD are presented as eight 32-bit values. [Table 132](#) is a field list of the basic TCD structure.

**Table 132. TCDn 32-bit memory structure**

eDMA offset	TCDn field	
0x1000+(32 x n)+0x0000	Source address (saddr)	
0x1000+(32 x n)+0x0004	Transfer attributes	Signed source address offset (soff)
0x1000+(32 x n)+0x0008	Inner minor byte count (nbytes)	
0x1000+(32 x n)+0x000C	Last source address adjustment (slast)	
0x1000+(32 x n)+0x0010	Destination address (daddr)	
0x1000+(32 x n)+0x0014	Current major iteration count (citer)	Signed destination address offset (doff)
0x1000 (32 x n) 0x0018	Last destination address adjustment / scatter-gather address (dlast_sga)	
0x1000+(32 x n)+0x001c	Beginning major iteration count (biter)	Channel control/status

[Figure 136](#) and [Table 133](#) define the fields of the TCDn structure.

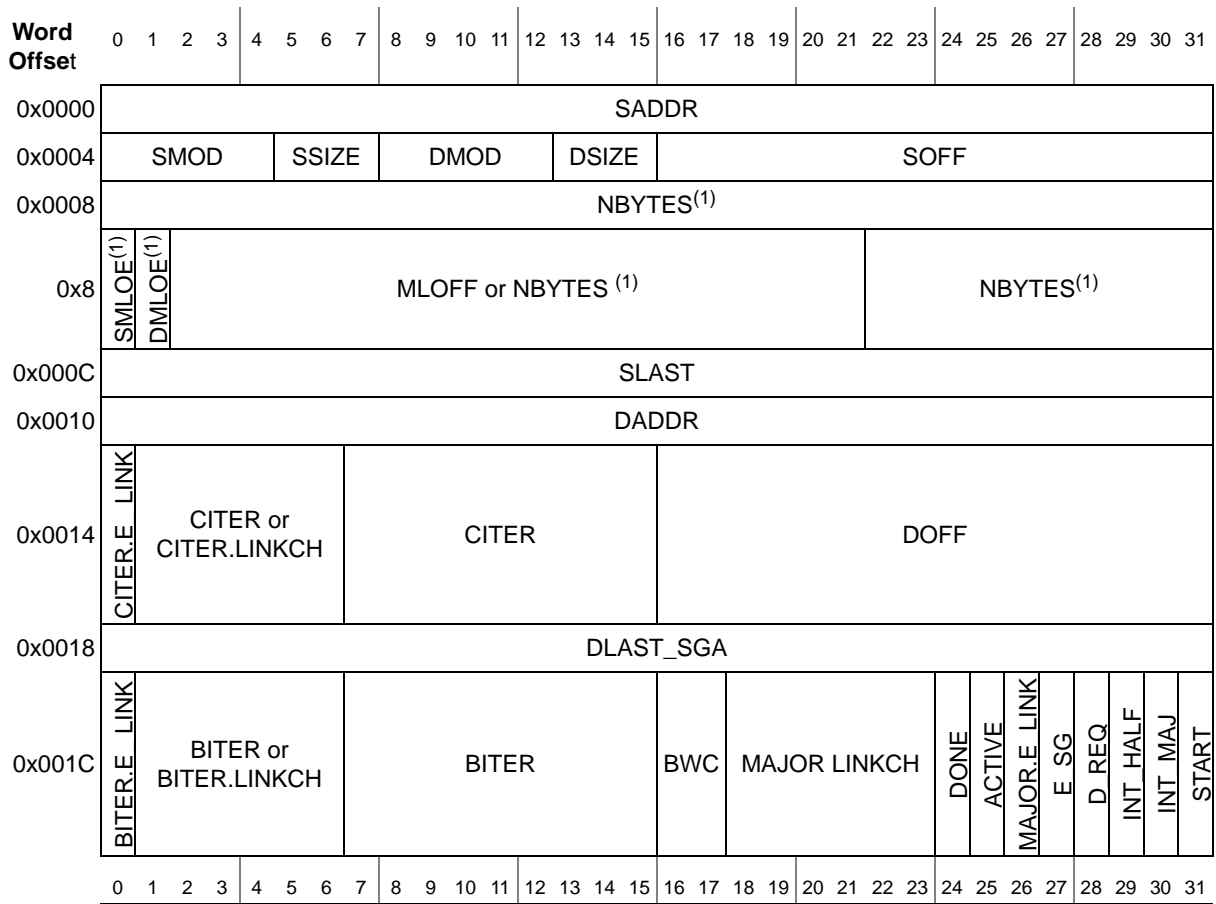


Figure 136. TCD structure

1. The fields implemented in Word 2 depend on whether EDMA\_CR(EMLM) is set to 0 or 1. See [Table 116](#).

**Note:** *The TCD structures for the eDMA channels shown in [Figure 136](#) are implemented in internal SRAM. These structures are not initialized at reset; therefore, all channel TCD parameters must be initialized by the application code before activating that channel.*

Table 133. TCDn field descriptions

Bits / Word Offset [n:n]	Name	Description
0–31 / 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 / 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. non-0 This value defines a specific address range that is specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.
37–39 / 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 Reserved 100 16-byte (32-bit, 4-beat, WRAP4 burst) 101 32-byte (32-bit, 8 beat, WRAP8 burst) 110 Reserved 111 Reserved The attempted specification of a reserved encoding causes a configuration error.
40–44 / 0x4 [8:12]	DMOD [0:4]	Destination address modulo. See the SMOD[0:5] definition.
45–47 / 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. See the SSIZE[0:2] definition.
48–63 / 0x4 [16:31]	SOFF [0:15]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64–95 / 0x8 [0:31]	NBYTES [0:31]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the DMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. After the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. Note: The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.

Table 133. TCDn field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
64–95 / 0x8 [0:31]	NBYTES <sup>(1)</sup> [0:31]	<p>Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed.</p> <p><b>Note:</b> The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 Gbyte transfer.</p>
64 0x8 [0]	SMLOE <sup>(1)</sup> 0	<p>Source minor loop offset enable This flag selects whether the minor loop offset is applied to the source address upon minor loop completion.</p> <p>0 The minor loop offset is not applied to the saddr. 1 The minor loop offset is applied to the saddr.</p>
65 0x8 [1]	DMLOE <sup>(1)</sup> 1	<p>Destination minor loop offset enable This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion.</p> <p>0 The minor loop offset is not applied to the daddr. 1 The minor loop offset is applied to the daddr.</p>
66–85 0x8 [2-21]	MLOFF or NBYTES <sup>(1)</sup> [0:19]	<p>Inner “minor” byte transfer count or Minor loop offset If both SMLOE and DMLOE are cleared, this field is part of the byte transfer count.</p> <p>If either SMLOE or DMLOE are set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop is completed.</p>
86–95 / 0x8 [22:31]	NBYTES <sup>(1)</sup>	<p>Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed.</p> <p><b>Note:</b> The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GByte transfer.</p>
96–127 / 0xC [0:31]	SLAST [0:31]	<p>Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.</p>

Table 133. TCDn field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
128–159 / 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 / 0x14 [0]	CITER.E_LINK	<p>Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking.</p> <p>0 The channel-to-channel linking is disabled.                      1 The channel-to-channel linking is enabled.</p> <p><b>Note:</b> This bit must be equal to the BITER.E_LINK bit otherwise a configuration error will be reported.</p>
161–166 / 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	<p>Current major iteration count or link channel number.</p> <p>If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then</p> <ul style="list-style-type: none"> <li>– No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field.</li> </ul> <p>Otherwise,</p> <ul style="list-style-type: none"> <li>– After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel's TCD.START bit.</li> </ul>
167–175 / 0x14 [7:15]	CITER [6:14]	<p>Current major iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field.</p> <p><b>Note:</b> When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field.</p> <p><b>Note:</b> If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p>
176–191 / 0x14 [16:31]	DOFF [0:15]	Destination address signed Offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Table 133. TCDn field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
192–223 / 0x18 [0:31]	DLAST_SGA [0:31]	<p>Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter-gather). If scatter-gather processing for the channel is disabled (TCD.E_SG = 0) then</p> <ul style="list-style-type: none"> <li>– Adjustment value added to the destination address at the completion of the outer major iteration count.</li> </ul> <p>This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure.</p> <p>Otherwise,</p> <ul style="list-style-type: none"> <li>– This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter-gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.</li> </ul>
224 / 0x1C [0]	BITER.E_LINK	<p>Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking.</p> <p>0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.</p> <p><b>Note:</b> When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.</p>
225–230 / 0x1C [1:6]	BITER [0:5] or BITER.LINKCH[0:5]	<p>Starting major iteration count or link channel number.</p> <p>If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then</p> <ul style="list-style-type: none"> <li>– No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field.</li> </ul> <p>Otherwise,</p> <ul style="list-style-type: none"> <li>– After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel's TCD.START bit.</li> </ul> <p><b>Note:</b> When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.</p>
231–239 / 0x1C [7:15]	BITER [6:14]	<p>Starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field.</p> <p><b>Note:</b> If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p>

Table 133. TCDn field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
240–241 / 0x1C [16:17]	BWC [0:1]	<p>Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR).</p> <p>00 No DMA engine stalls                      01 Reserved                      10 DMA engine stalls for 4 cycles after each r/w                      11 DMA engine stalls for 8 cycles after each r/w</p>
242–247 / 0x1C [18:23]	MAJOR.LINKCH [0:5]	<p>Link channel number.</p> <p>If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then,</p> <ul style="list-style-type: none"> <li>– No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted.</li> </ul> <p>Otherwise</p> <ul style="list-style-type: none"> <li>– After the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.</li> </ul>
248 / 0x1C [24]	DONE	<p>Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the DMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the DMA engine has begun processing the channel, not when the first data transfer occurs).</p> <p><b>Note:</b> This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.</p>
249 / 0x1C [25]	ACTIVE	<p>Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.</p>
250 / 0x1C [26]	MAJOR.E_LINK	<p>Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel.</p> <p><b>Note:</b> To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set.</p> <p>0 The channel-to-channel linking is disabled.                      1 The channel-to-channel linking is enabled.</p>

Table 133. TCDn field descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
251 / 0x1C [27]	E_SG	<p>Enable scatter-gather processing. As the channel completes the outer major loop, this flag enables scatter-gather processing in the current channel. If enabled, the DMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory.</p> <p><b>Note:</b> To support the dynamic scatter-gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set.</p> <p>0 The current channel's TCD is normal format.                      1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.</p>
252 / 0x1C [28]	D_REQ	<p>Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQRL bit when the current major iteration count reaches zero.</p> <p>0 The channel's EDMA_ERQRL bit is not affected.                      1 The channel's EDMA_ERQRL bit is cleared when the outer major loop is complete.</p>
253 / 0x1C [29]	INT_HALF	<p>Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQRL when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER &gt;&gt; 1)). This halfway point interrupt request is provided to support double-buffered (aka ping-pong) schemes, or other types of data movement where the processor needs an early indication of the transfer's progress. CITER = BITER = 1 with INT_HALF enabled will generate an interrupt as it satisfies the equation (CITER == (BITER &gt;&gt; 1)) after a single activation.</p> <p>0 The half-point interrupt is disabled.                      1 The half-point interrupt is enabled.</p>
254 / 0x1C [30]	INT_MAJ	<p>Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQRL when the current major iteration count reaches zero.</p> <p>0 The end-of-major loop interrupt is disabled.                      1 The end-of-major loop interrupt is enabled.</p>
255 / 0x1C [31]	START	<p>Channel start. If this flag is set the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution.</p> <p>0 The channel is not explicitly started.                      1 The channel is explicitly started via a software initiated service request.</p>

1. The fields implemented at 0x8 depend on whether EDMA\_CR(EMLM) is set to 0 or 1. Refer to [Table 116](#).

## 16.4 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA block.



The eDMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. The DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
  - Address path: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA\_CPR $n$ [ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution. When another channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCD $n$ .{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCD $n$ .CITER field, and a possible fetch of the next TCD $n$  from memory as part of a scatter-gather operation.
  - Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output. The address and data path modules directly support the two-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the second stage of the pipeline (the data phase).
  - Program model/channel arbitration: This module implements the first section of eDMA's programming model and also the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the control logic).
  - Control: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved. A minor loop interaction is defined as the number of bytes to transfer ( $n$ bytes) divided by the transfer size. Transfer size is defined as:
 

```
if (SSIZE < DSIZE)
transfer size = destination transfer size (# of bytes)
else
transfer size = source transfer size (# of bytes)
```

 Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST,

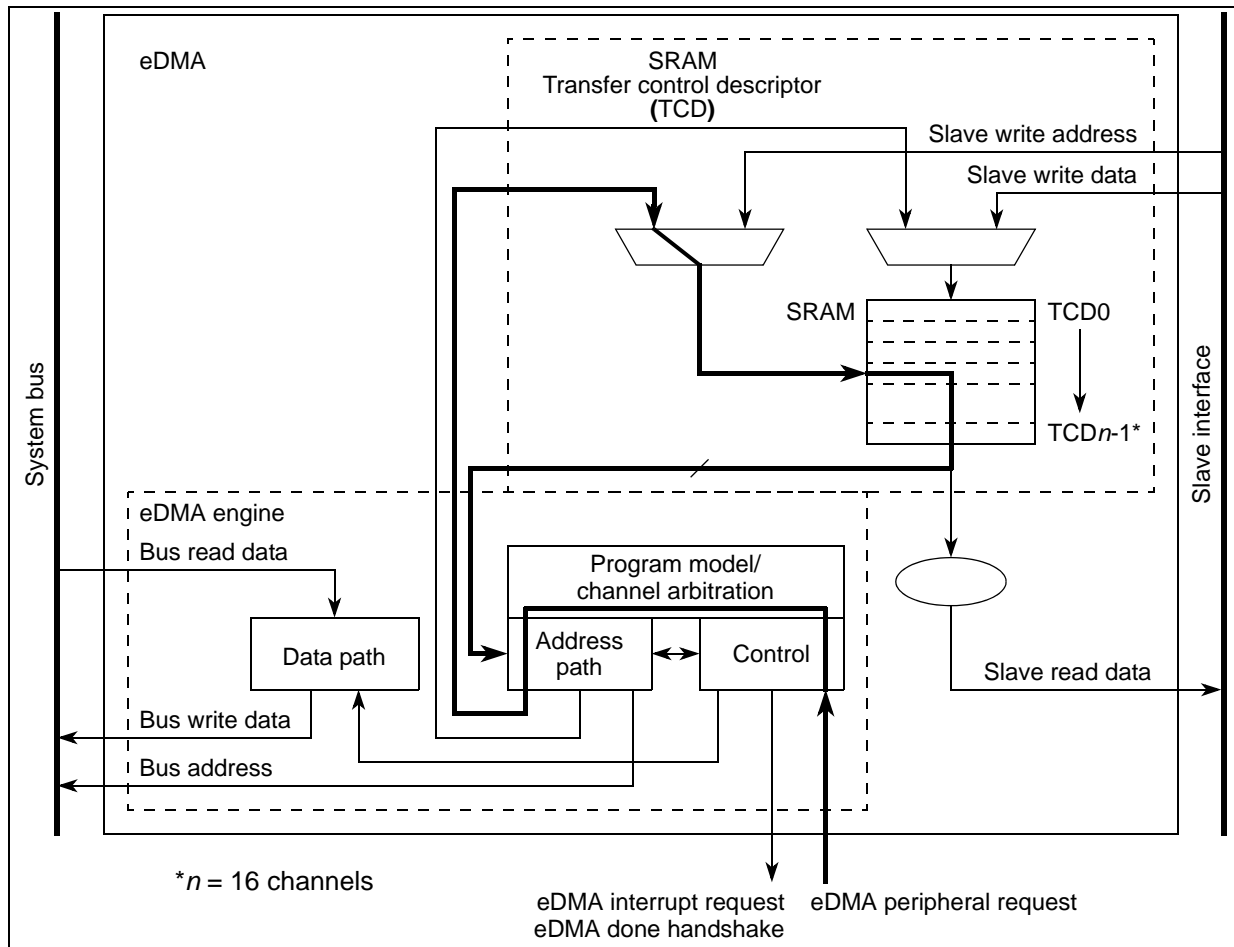
SLAST, CITER, BITER, DONE, D\_REQ, INT\_MAJ, MAJOR\_LNKCH, and INT\_HALF.

For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  - Memory controller: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.
  - Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

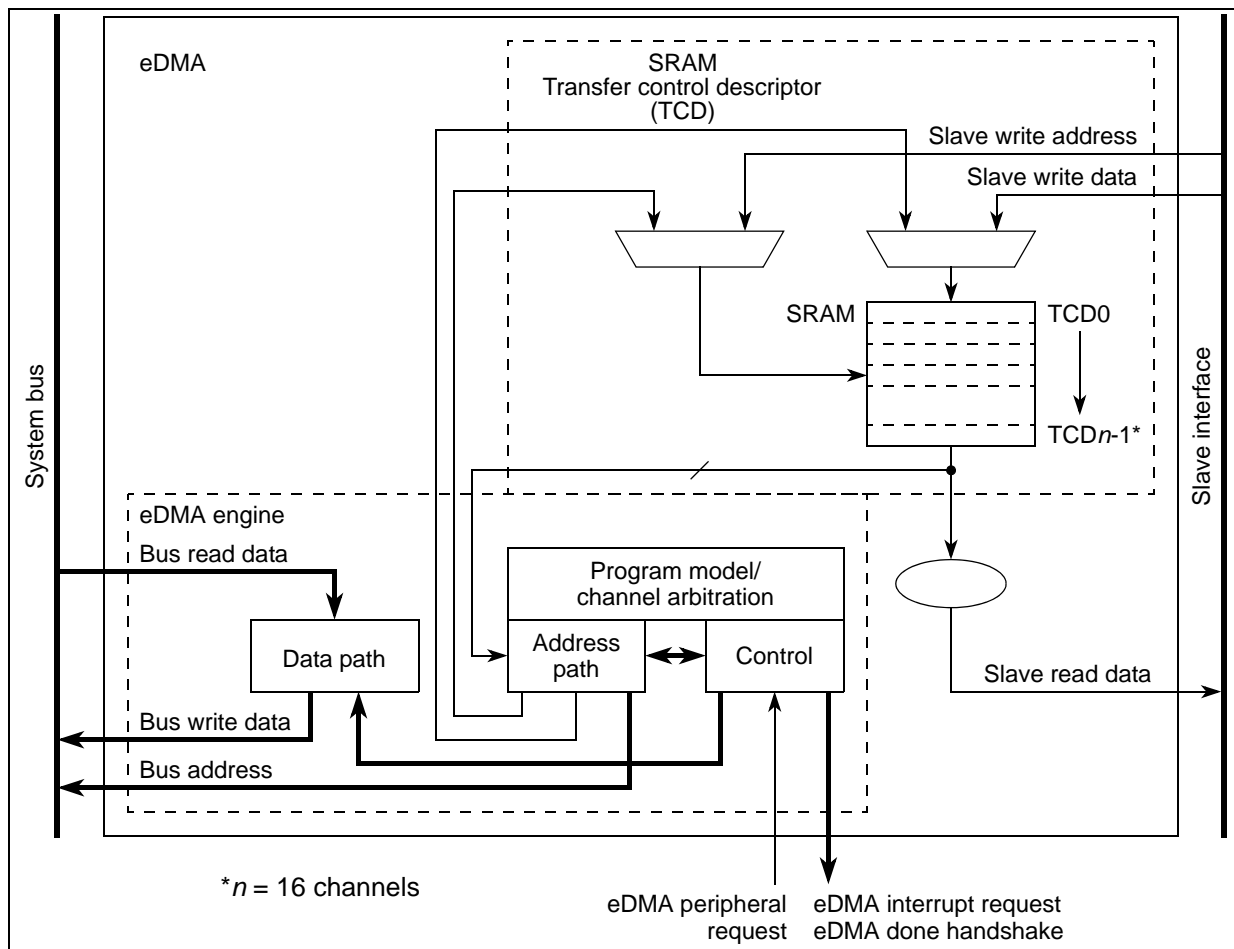
### 16.4.1 eDMA basic data flow

The eDMA transfers data based on a two-deep, nested flow. The basic flow of a data transfer can be partitioned into three segments. As shown in [Figure 137](#), the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel  $n$ . Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed to through the DMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.



**Figure 137. eDMA operation, part 1**

In the second part of the basic data flow as shown in [Figure 138](#), the modules associated with the data transfer (address path, data path, and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA done handshake signal is asserted at the end of the minor byte count transfer.



**Figure 138. eDMA operation, part 2**

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD; for example, SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter-gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in [Figure 139](#).

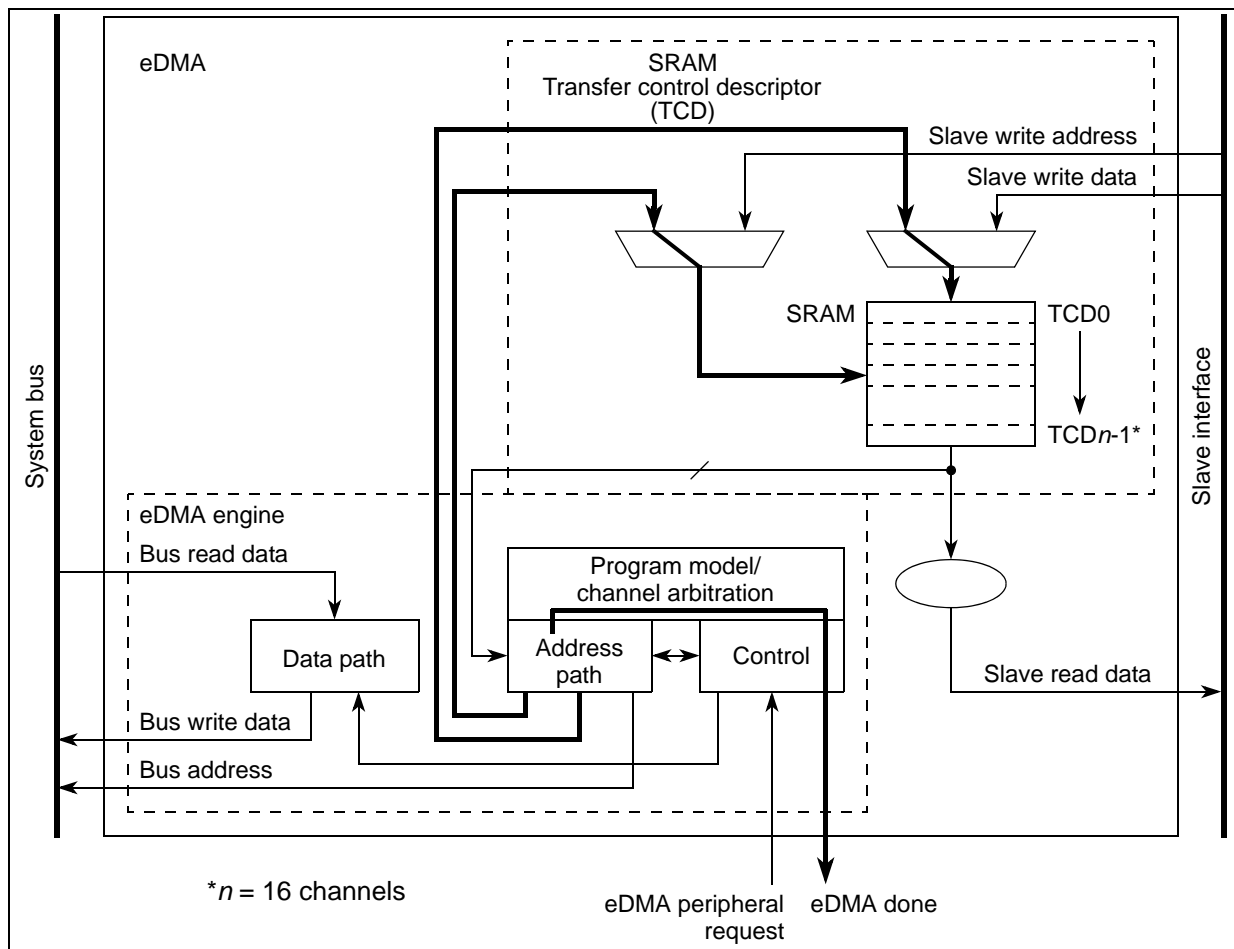


Figure 139. eDMA operation, part 3

## 16.5 Initialization / application information

### 16.5.1 eDMA initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA\_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA\_CPR<sub>n</sub> registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA\_EEIRL and/or EDMA\_EEIRH registers if desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA\_ERQRH and/or EDMA\_ERQRL registers.
6. Request channel service by software (setting the TCD.START bit) or by hardware (slave device asserting its DMA peripheral request signal).

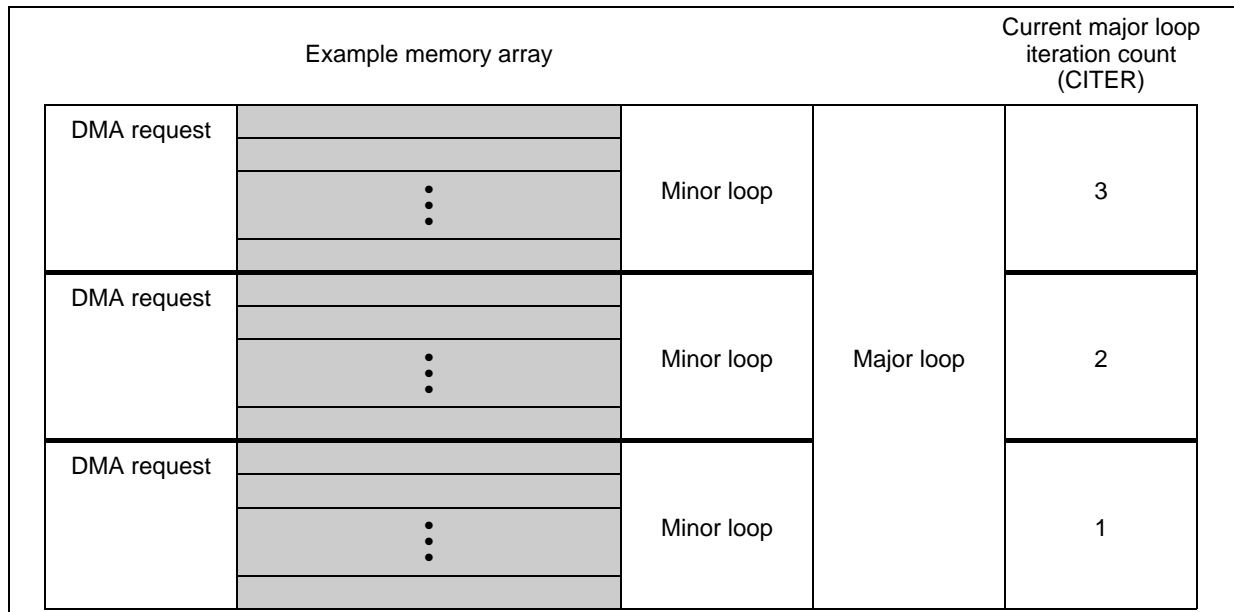
After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine will read

the entire TCD, including the primary transfer control parameter shown in [Table 134](#), for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the DMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed; for example, interrupts, major loop channel linking, and scatter-gather operations, if enabled.

**Table 134. TCD primary control and status fields**

TCD field name	Description
START	Control bit to start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for throttling bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

[Figure 140](#) shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).



**Figure 140. Example of multiple loop iterations**

Figure 141 lists the memory array terms and how the TCD settings interrelate.

xADDR: (Starting address)	xSIZE: (Size of one data transfer)	Minor loop (NBYTES in minor loop, often the same value as xSIZE)	Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE)  Each DMA source (S) and destination (D) has its own: <ul style="list-style-type: none"> <li>• Address (xADDR)</li> <li>• Size (xSIZE)</li> <li>• Offset (xOFF)</li> <li>• Modulo (xMOD)</li> <li>• Last address adjustment (xLAST) where x = S or D</li> </ul> Peripheral queues typically have size and offset equal to NBYTES
⋮	⋮	Minor loop	
xLAST: Number of bytes added to current address after major loop (typically used to loop back)	⋮	Last minor loop	

**Figure 141. Memory array terms**

### 16.5.2 DMA programming errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per-channel basis with the exception of channel-priority error, or EDMA\_ESR[CPE].

For all error types other than channel-priority errors, the channel number causing the error is recorded in the EDMA\_ESR. If the error source is not removed before the next activation of the problem channel, the error will be detected and recorded again.

If priority levels are not unique, the highest (channel) priority that has an active request is selected, but the lowest numbered (channel) with that priority is selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

### 16.5.3 DMA request assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in [Table 135](#). The source column is written in C language syntax. The syntax is `module_instance.register[bit]`.

**Table 135. DMA Request Summary for eDMA**

DMA Request	Channel	Source	Description
DMA_MUX_CHCONFIG0_SOURCE	0	DMA_MUX.CHCONFIG0[SOURCE]	DMA MUX channel 0 source
DMA_MUX_CHCONFIG1_SOURCE	1	DMA_MUX.CHCONFIG1[SOURCE]	DMA MUX channel 1 source
DMA_MUX_CHCONFIG2_SOURCE	2	DMA_MUX.CHCONFIG2[SOURCE]	DMA MUX channel 2 source
DMA_MUX_CHCONFIG3_SOURCE	3	DMA_MUX.CHCONFIG3[SOURCE]	DMA MUX channel 3 source
DMA_MUX_CHCONFIG4_SOURCE	4	DMA_MUX.CHCONFIG4[SOURCE]	DMA MUX channel 4 source
DMA_MUX_CHCONFIG5_SOURCE	5	DMA_MUX.CHCONFIG5[SOURCE]	DMA MUX channel 5 source
DMA_MUX_CHCONFIG6_SOURCE	6	DMA_MUX.CHCONFIG6[SOURCE]	DMA MUX channel 6 source
DMA_MUX_CHCONFIG7_SOURCE	7	DMA_MUX.CHCONFIG7[SOURCE]	DMA MUX channel 7 source
DMA_MUX_CHCONFIG8_SOURCE	8	DMA_MUX.CHCONFIG8[SOURCE]	DMA MUX channel 8 source
DMA_MUX_CHCONFIG9_SOURCE	9	DMA_MUX.CHCONFIG9[SOURCE]	DMA MUX channel 9 source
DMA_MUX_CHCONFIG10_SOURCE	10	DMA_MUX.CHCONFIG10[SOURCE]	DMA MUX channel 10 source
DMA_MUX_CHCONFIG11_SOURCE	11	DMA_MUX.CHCONFIG11[SOURCE]	DMA MUX channel 11 source
DMA_MUX_CHCONFIG12_SOURCE	12	DMA_MUX.CHCONFIG12[SOURCE]	DMA MUX channel 12 source
DMA_MUX_CHCONFIG13_SOURCE	13	DMA_MUX.CHCONFIG13[SOURCE]	DMA MUX channel 13 source
DMA_MUX_CHCONFIG14_SOURCE	14	DMA_MUX.CHCONFIG14[SOURCE]	DMA MUX channel 14 source
DMA_MUX_CHCONFIG15_SOURCE	15	DMA_MUX.CHCONFIG15[SOURCE]	DMA MUX channel 15 source



## 16.5.4 DMA arbitration mode considerations

### 16.5.4.1 Fixed-channel arbitration

In this mode, the channel service request from the highest priority channel is selected to execute. Preemption is available in this scenario only.

### 16.5.4.2 Round-robin channel arbitration

In this mode, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the assigned channel priority levels.

## 16.5.5 DMA transfer

### 16.5.5.1 Single request

To perform a simple transfer of  $n$  bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer will begin after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD.DONE bit will be set and an interrupt will be generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

```
TCD.CITER = TCD.BITER = 1
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = -16
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA = -16
TCD.INT_MAJ = 1
TCD.START = 1 (Must be written last after all other fields have been initialized)
All other TCD fields = 0
```

This would generate the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
  - a) read\_byte(0x1000), read\_byte(0x1001), read\_byte(0x1002), read\_byte(0x1003)
  - b) write\_word(0x2000) → first iteration of the minor loop
  - c) read\_byte(0x1004), read\_byte(0x1005), read\_byte(0x1006), read\_byte(0x1007)
  - d) write\_word(0x2004) → second iteration of the minor loop
  - e) read\_byte(0x1008), read\_byte(0x1009), read\_byte(0x100a), read\_byte(0x100b)
  - f) write\_word(0x2008) → third iteration of the minor loop
  - g) read\_byte(0x100c), read\_byte(0x100d), read\_byte(0x100e), read\_byte(0x100f)
  - h) write\_word(0x200c) → last iteration of the minor loop → major loop complete
6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA\_IRQR<sub>n</sub> = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

### 16.5.5.2 Multiple requests

The next example is the same as previous, excepting transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA\_ERQR, channel service requests are initiated by the slave device (ERQR should be set after TCD). Note that TCD.START = 0.

```

TCD.CITER = TCD.BITER = 2
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = -32
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA = -32
TCD.INT_MAJ = 1
TCD.START = 0 (Must be written last after all other fields have been initialized)
All other TCD fields = 0

```

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
  - a) read\_byte(0x1000), read\_byte(0x1001), read\_byte(0x1002), read\_byte(0x1003)
  - b) write\_word(0x2000) → first iteration of the minor loop
  - c) read\_byte(0x1004), read\_byte(0x1005), read\_byte(0x1006), read\_byte(0x1007)
  - d) write\_word(0x2004) → second iteration of the minor loop
  - e) read\_byte(0x1008), read\_byte(0x1009), read\_byte(0x100a), read\_byte(0x100b)
  - f) write\_word(0x2008) → third iteration of the minor loop
  - g) read\_byte(0x100c), read\_byte(0x100d), read\_byte(0x100e), read\_byte(0x100f)
  - h) write\_word(0x200c) → last iteration of the minor loop
6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
7. eDMA engine writes: TCD.ACTIVE = 0.
8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
  - a) read\_byte(0x1010), read\_byte(0x1011), read\_byte(0x1012), read\_byte(0x1013)
  - b) write\_word(0x2010) → first iteration of the minor loop
  - c) read\_byte(0x1014), read\_byte(0x1015), read\_byte(0x1016), read\_byte(0x1017)
  - d) write\_word(0x2014) → second iteration of the minor loop
  - e) read\_byte(0x1018), read\_byte(0x1019), read\_byte(0x101a), read\_byte(0x101b)
  - f) write\_word(0x2018) → third iteration of the minor loop
  - g) read\_byte(0x101c), read\_byte(0x101d), read\_byte(0x101e), read\_byte(0x101f)
  - h) write\_word(0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA\_IRQR<sub>n</sub> = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

### 16.5.5.3 Modulo feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit bitfield for both the source and destination in the TCD and specifies which lower address bits are allowed to increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 136 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2<sup>4</sup> byte (16-byte) size queue.

**Table 136. Modulo Feature Example**

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

### 16.5.6 TCD status

#### 16.5.6.1 Minor loop complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method may be extracted from the sequence below. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a 1. Polling the TCD.ACTIVE bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (channel service request via software).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer’s model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (channel service request via hardware).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution, regardless of how the channel was activated.

#### 16.5.6.2 Active channel TCD reads

The eDMA will read back the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

#### 16.5.6.3 Preemption status

Preemption is available only when fixed arbitration is selected for channel-arbitration mode. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed-channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

#### 16.5.7 Channel linking

Channel linking (or chaining) is a mechanism in which one channel sets the TCD.START bit of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E\_LINK field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

```
TCD.CITER.E_LINK = 1
TCD.CITER.LINKCH = 0xC
TCD.CITER value = 0x4
TCD.MAJOR.E_LINK = 1
TCD.MAJOR.LINKCH = 0x7
```

will execute as:

1. Minor loop done → set channel 12 TCD.START bit
2. Minor loop done → set channel 12 TCD.START bit
3. Minor loop done → set channel 12 TCD.START bit
4. Minor loop done, major loop done → set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E\_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E\_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

*Note:* After configuration, the TCD.CITER.E\_LINK bit and the TCD.BITER.E\_LINK bit must be equal or a configuration error will be reported. The CITER and BITER vector widths must be equal to calculate the major loop, halfway done interrupt point.

Table 137 summarizes how a DMA channel can link to another DMA channel, i.e, use another channel's TCD, at the end of a loop.

**Table 137. Channel linking parameters**

Desired Link Behavior	TCD Control Field Name	Description
Link at end of minor loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration).
	citer.linkch	Link channel number when linking at end of minor loop (current iteration).
Link at end of major loop	major.e_link	Enable channel-to-channel linking on major loop completion.
	major.linkch	Link channel number when linking at end of major loop.

## 16.5.8 Dynamic programming

### 16.5.8.1 Dynamic channel linking

Dynamic channel linking is the process of setting the TCD.major.e\_link bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.major.e\_link bit at the same time the eDMA engine is retiring the channel. The TCD.major.e\_link would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The coherency model in Table 138 is recommended when executing a dynamic channel link request.

**Table 138. Coherency model for a dynamic channel link request**

Step	Action
1	Write 1b to the TCD.major.e_link bit.
2	Read back the TCD.major.e_link bit.
3	Test the TCD.major.e_link request status: – If TCD.major.e_link = 1b, the dynamic link attempt was successful. – If TCD.major.e_link = 0b, the attempted dynamic link did not succeed (the channel was already retiring).

For this request, the TCD local memory controller forces the TCD.major.e\_link bit to zero on any writes to a channel's TCD.word7 after that channel's TCD.done bit is set, indicating the major loop is complete.

*Note:* The user must clear the TCD.done bit before writing the TCD.major.e\_link bit. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

### 16.5.8.2 Dynamic scatter/gather

Dynamic scatter/gather is the process of setting the TCD.e\_sg bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic scatter/gather operation by enabling the TCD.e\_sg bit at the same time the eDMA engine is retiring the channel. The TCD.e\_sg would be set in the programmer's model, but it would be unclear whether the actual scatter/gather request was honored before the channel retired. Two methods for this coherency model are shown in the following subsections.

Method 1 has the advantage of reading the major.linkch field and the e\_sg bit with a single read. For both dynamic channel linking and scatter/gather requests, the TCD local memory controller forces the TCD.major.e\_link and TCD.e\_sg bits to zero on any writes to a channel's TCD.word7 if that channel's TCD.done bit is set indicating the major loop is complete.

*Note:* The user must clear the TCD.done bit before writing the TCD.major.e\_link or TCD.e\_sg bits. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

#### 16.5.8.2.1 Method 1 (channel not using major loop channel linking)

For a channel not using major loop channel linking, the coherency model in [Table 139](#) may be used for a dynamic scatter/gather request.

When the TCD.major.e\_link bit is zero, the TCD.major.linkch field is not used by the eDMA. In this case, the TCD.major.linkch bits may be used for other purposes. This method uses the TCD.major.linkch field as a TCD identification (ID).

**Table 139. Coherency model for method 1**

Step	Action
1	When the descriptors are built, write a unique TCD ID in the TCD.major.linkch field for each TCD associated with a channel using dynamic scatter/gather.
2	Write 1b to the TCD.d_req bit. Note: Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offset value.
3	Write the TCD.dlast_sga field with the scatter/gather address.
4	Write 1b to the TCD.e_sg bit.
5	Read back the 16 bit TCD control/status field.
6	Test the TCD.e_sg request status and TCD.major.linkch value: <ul style="list-style-type: none"> <li>– If e_sg = 1b, the dynamic link attempt was successful.</li> <li>– If e_sg = 0b and the major.linkch (ID) did not change, the attempted dynamic link did not succeed (the channel was already retiring).</li> <li>– If e_sg = 0b and the major.linkch (ID) changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).</li> </ul>

**16.5.8.2.2 Method 12 (channel using major loop linking)**

For a channel using major loop channel linking, the coherency model in [Table 140](#) may be used for a dynamic scatter/gather request. This method uses the TCD.dlast\_sga field as a TCD identification (ID).

**Table 140. Coherency model for method 2**

Step	Action
1	Write 1b to the TCD.d_req bit. Note: Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offset value.
2	Write the TCD.dlast_sga field with the scatter/gather address.
3	Write 1b to the TCD.e_sg bit.
4	Read back the TCD.e_sg bit.
5	Test the TCD.e_sg request status: <ul style="list-style-type: none"> <li>– If e_sg = 1b, the dynamic link attempt was successful.</li> <li>– If e_sg = 0b, read the 32 bit TCD dlast_sga field.</li> <li>– If e_sg = 0b and the dlast_sga did not change, the attempted dynamic link did not succeed (the channel was already retiring).</li> <li>– If e_sg = 0b and the dlast_sga changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).</li> </ul>



# 17 eDMA Channel Multiplexer (DMA\_MUX)

## 17.1 Introduction

The eDMA channel multiplexer (DMA\_MUX) allows the routing of 59 DMA sources (slots) to 16 eDMA channels. This is illustrated in [Figure 142](#).

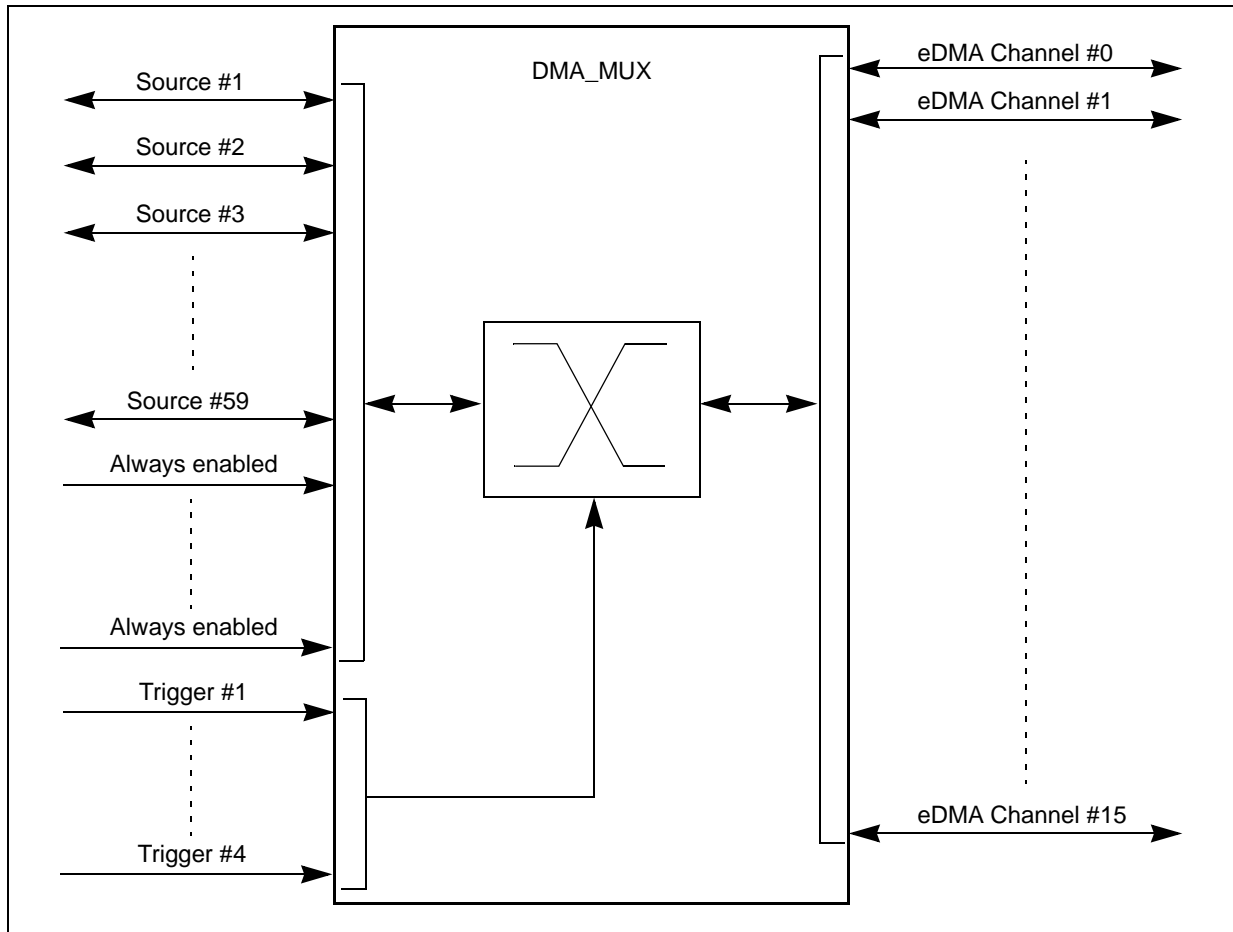


Figure 142. DMA\_MUX block diagram

## 17.2 Features

The DMA\_MUX has these major features:

- 16 independently selectable eDMA channel routers
  - 4 channels with normal or periodic triggering capability
  - 12 channels with normal capability
- Capability to assign each channel router to 1 of 59 possible peripheral DMA sources, 4 always enabled sources or 1 always disabled source
- 3 modes of operation:
  - Disabled
  - Normal
  - Periodic Trigger

## 17.3 Modes of operation

The following operation modes are available:

- Disabled Mode — In this mode, the eDMA channel is disabled. Since disabling and enabling of eDMA channels is done primarily via the eDMA configuration registers, this mode is used mainly as the reset state for a eDMA channel in the DMA\_MUX. It may also be used to temporarily suspend a eDMA channel while reconfiguration of the system takes place (for example, changing the period of a eDMA trigger).
- Normal Mode — In this mode, a eDMA source (such as DSPI\_0\_TX or DSPI\_0\_RX example) is routed directly to the specified eDMA channel. The operation of the DMA\_MUX in this mode is completely transparent to the system.
- Periodic Trigger Mode — In this mode, a eDMA source may only request a eDMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. The period is configured in the registers of the Periodic Interrupt Timer (PIT).

eDMA channels 0–3 may be used in all three modes, but channels 4–15 may only be configured to disabled or normal mode.

## 17.4 External signal description

The DMA\_MUX has no external pins.

## 17.5 Memory map and register definition

*Table 141* shows the memory map for the DMA\_MUX. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMA\_MUX.

**Table 141. DMA\_MUX memory map**

Base address: 0xFFFD_C000		
Address offset	Register	Location
0x0	Channel #0 Configuration (CHCONFIG0)	<a href="#">on page 347</a>
0x1	Channel #1 Configuration (CHCONFIG1)	<a href="#">on page 347</a>
...	...	...
0xF	Channel #15 Configuration (CHCONFIG15)	<a href="#">on page 347</a>

All registers are accessible via 8, 16 or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, CHCONFIG0 through CHCONFIG3 are accessible by a 32-bit READ/WRITE to address 'Base + 0x00', but performing a 32-bit access to address 'Base + 0x01' is illegal.

### 17.5.1 Channel configuration registers (CHCONFIG<sub>n</sub>)

Each of the total of 16 eDMA channels can be independently enabled/disabled and associated with 1 of the 28 peripheral eDMA sources + 1 of the 4 always enabled eDMA sources in the system.

Offset: 0x0 + n (16 registers)

Access: User read/write



**Figure 143. Channel Configuration Registers (CHCONFIG<sub>n</sub>)**

**Table 142. CHCONFIG<sub>n</sub> field descriptions**

Field	Description
ENBL	eDMA Channel Enable ENBL enables the eDMA channel. 0 eDMA channel is disabled. This mode is primarily used during configuration of the DMA_MUX. The eDMA has separate channel enables/disables, which should be used to disable or reconfigure a eDMA channel. 1 eDMA channel is enabled
TRIG	eDMA Channel Trigger Enable (for triggered channels only) TRIG enables the periodic trigger capability for the eDMA channel. 0 Periodic triggering is disabled. If periodic triggering is disabled, and the ENBL bit is set, the DMA_MUX will simply route the specified source to the eDMA channel. 1 Triggering is enabled
SOURCE	eDMA Channel Source (slot) SOURCE specifies which eDMA source, if any, is routed to a particular eDMA channel. Please see <a href="#">Table 144</a> for DMA_MUX inputs mapping.

**Table 143. Channel and trigger enabling**

ENBL	TRIG	Function	Mode
0	X	eDMA channel is disabled	Disabled Mode
1	0	eDMA channel is enabled with no triggering (transparent)	Normal Mode
1	1	eDMA channel is enabled with triggering	Periodic Trigger Mode

*Note:* Setting multiple CHCONFIG registers with the same Source value results in unpredictable behavior.

*Note:* Before changing the trigger or source settings a eDMA channel must be disabled via the CHCONFIGn[ENBL] bit.

## 17.6 DMA\_MUX inputs

### 17.6.1 DMA\_MUX peripheral sources

**Table 144. eDMA channel mapping**

DMA_MUX channel	Module	eDMA requesting module	DMA_MUX input #
0	—	Always disabled	—
1	DSPI 0	DSPI_0 TX	DMA_MUX Source #1
2	DSPI 0	DSPI_0 RX	DMA_MUX Source #2
3	DSPI 1	DSPI_1 TX	DMA_MUX Source #3
4	DSPI 1	DSPI_1 RX	DMA_MUX Source #4
5	DSPI 2	DSPI_2 TX	DMA_MUX Source #5
6	DSPI 2	DSPI_2 RX	DMA_MUX Source #6
7	DSPI 3	DSPI_3 TX	DMA_MUX Source #7
8	DSPI 3	DSPI_3 RX	DMA_MUX Source #8
9	DSPI 4	DSPI_4 TX	DMA_MUX Source #9
10	DSPI 4	DSPI_4 RX	DMA_MUX Source #10
11	DSPI 5	DSPI_5 TX	DMA_MUX Source #11
12	DSPI 5	DSPI_5 RX	DMA_MUX Source #12
13	—	—	DMA_MUX Source #13
14	—	—	DMA_MUX Source #14
15	—	—	DMA_MUX Source #15
16	—	—	DMA_MUX Source #16
17	eMIOS 0	EMIOS0_CH0	DMA_MUX Source #17
18	eMIOS 0	EMIOS0_CH1	DMA_MUX Source #18
19	eMIOS 0	EMIOS0_CH9	DMA_MUX Source #19
20	eMIOS 0	EMIOS0_CH18	DMA_MUX Source #20

Table 144. eDMA channel mapping (continued)

DMA_MUX channel	Module	eDMA requesting module	DMA_MUX input #
21	eMIOS 0	EMIOS0_CH25	DMA_MUX Source #21
22	eMIOS 0	EMIOS0_CH26	DMA_MUX Source #22
23	eMIOS 1	EMIOS1_CH0	DMA_MUX Source #23
24	eMIOS 1	EMIOS1_CH9	DMA_MUX Source #24
25	eMIOS 1	EMIOS1_CH17	DMA_MUX Source #25
26	eMIOS 1	EMIOS1_CH18	DMA_MUX Source #26
27	eMIOS 1	EMIOS1_CH25	DMA_MUX Source #27
28	eMIOS 1	EMIOS1_CH26	DMA_MUX Source #28
29	ADC 0	ADC0_EOC	DMA_MUX Source #29
30	ADC 1	ADC1_EOC	DMA_MUX Source #30
31	I <sup>2</sup> C	IIC_RX	DMA_MUX Source #31
32	I <sup>2</sup> C	IIC_TX	DMA_MUX Source #32
33	LINFLEX 0	LINFLEX0_RX	DMA_MUX Source #33
34	LINFLEX 0	LINFLEX0_TX	DMA_MUX Source #34
35	LINFLEX 1	LINFLEX1_RX	DMA_MUX Source #35
36	LINFLEX 1	LINFLEX1_TX	DMA_MUX Source #36
37	—	—	DMA_MUX Source #37
38	—	—	DMA_MUX Source #38
39	—	—	DMA_MUX Source #39
40	—	—	DMA_MUX Source #40
41	—	—	DMA_MUX Source #41
42	—	—	DMA_MUX Source #42
43	—	—	DMA_MUX Source #43
44	—	—	DMA_MUX Source #44
45	—	—	DMA_MUX Source #45
46	—	—	DMA_MUX Source #46
47	—	—	DMA_MUX Source #47
48	—	—	DMA_MUX Source #48
49	—	—	DMA_MUX Source #49
50	—	—	DMA_MUX Source #50
51	—	—	DMA_MUX Source #51
52	—	—	DMA_MUX Source #52
53	—	—	DMA_MUX Source #53
54	—	—	DMA_MUX Source #54
55	—	—	DMA_MUX Source #55

**Table 144. eDMA channel mapping (continued)**

DMA_MUX channel	Module	eDMA requesting module	DMA_MUX input #
56	—	—	DMA_MUX Source #56
57	—	—	DMA_MUX Source #57
58	—	—	DMA_MUX Source #58
59	—	—	DMA_MUX Source #59
60	—	ALWAYS ENABLED	DMA_MUX Source #60
61	—	ALWAYS ENABLED	DMA_MUX Source #61
62	—	ALWAYS ENABLED	DMA_MUX Source #62
63	—	ALWAYS ENABLED	DMA_MUX Source #63

**17.6.2 DMA\_MUX periodic trigger inputs**

**Table 145. DMA\_MUX periodic trigger inputs**

DMA_MUX trigger input	PIT channel
Trigger #1	PIT0
Trigger #2	PIT1
Trigger #3	PIT4
Trigger #4	PIT5

**17.7 Functional description**

The primary purpose of the DMA\_MUX is to provide flexibility in the system’s use of the available eDMA channels. As such, configuration of the DMA\_MUX is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in [Section 17.8.2: Enabling and configuring sources](#), is followed, the configuration of the DMA\_MUX may be changed during the normal operation of the system.

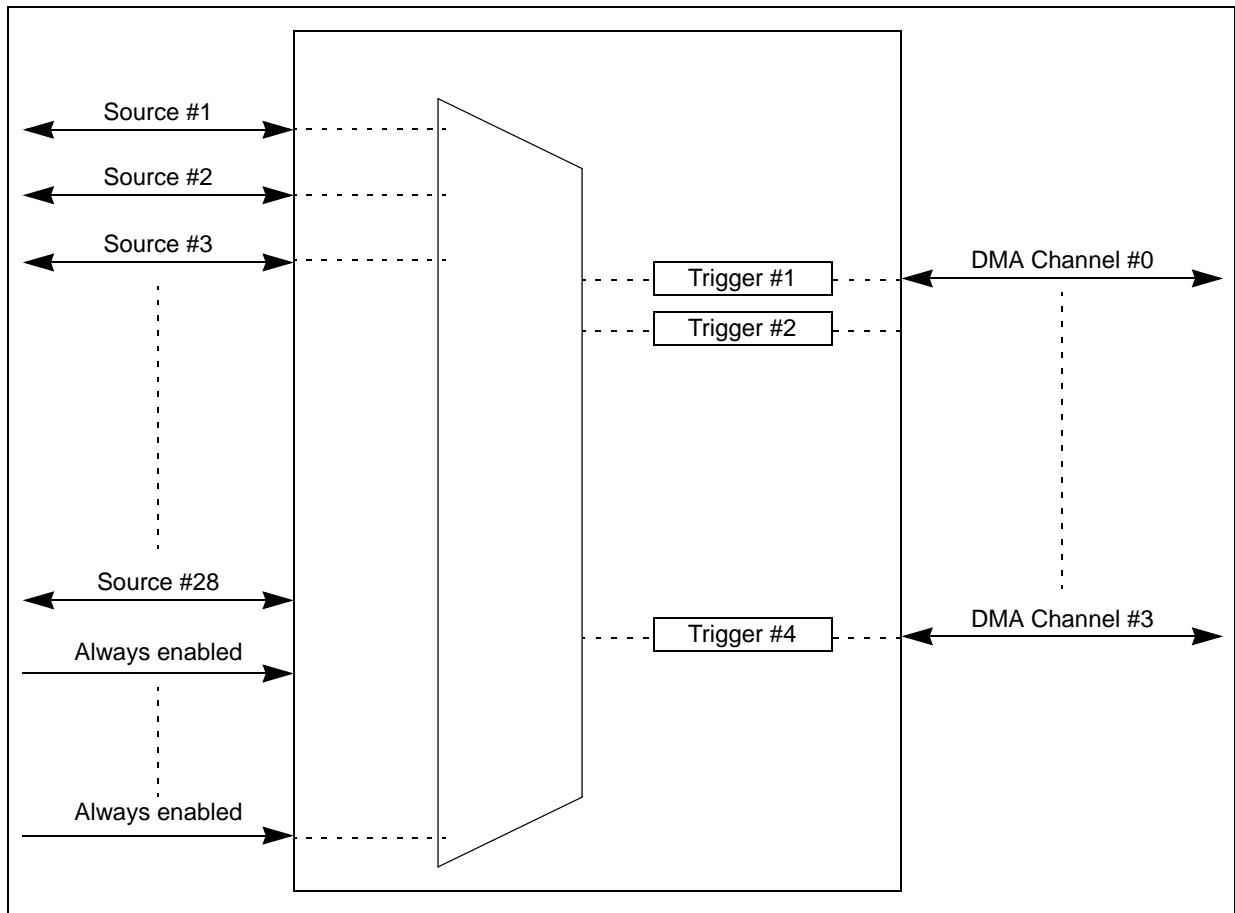
Functionally, the DMA\_MUX channels may be divided into two classes: Channels, which implement the normal routing functionality plus periodic triggering capability, and channels, which implement only the normal routing functionality.

**17.7.1 eDMA channels with periodic triggering capability**

Besides the normal routing functionality, the first four channels of the DMA\_MUX provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the periodic interrupt timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please refer to the periodic interrupt timer chapter of the reference manual for more information on this topic.

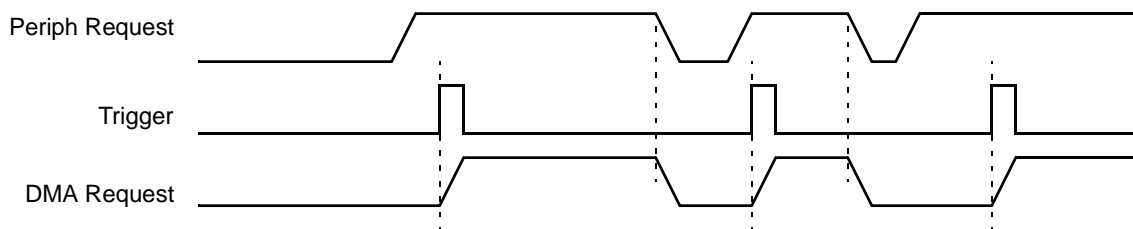
*Note: Because of the dynamic nature of the system (such as eDMA channel priorities, bus arbitration, or interrupt service routine lengths), the number of clock cycles between a trigger and the actual eDMA transfer cannot be guaranteed.*





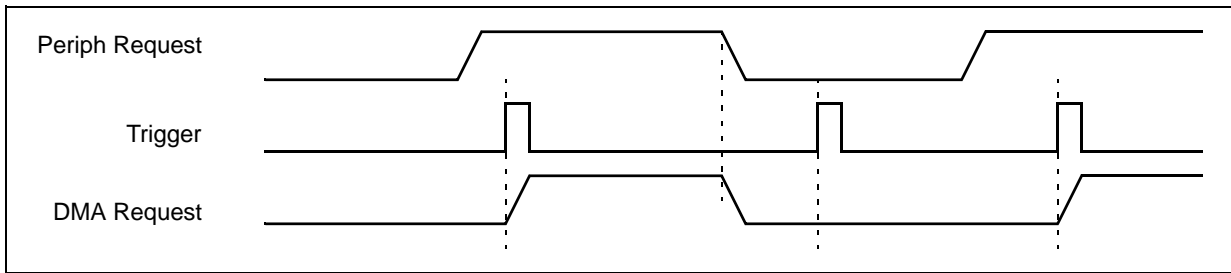
**Figure 144. DMA\_MUX channel 0–3 block diagram**

The eDMA channel triggering capability allows the system to “schedule” regular eDMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the eDMA until a trigger event has been seen. This is illustrated in [Figure 145](#).



**Figure 145. DMA\_MUX channel triggering: Normal operation**

Once the eDMA request has been serviced, the peripheral will negate its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that triggered will be ignored. This situation is illustrated in [Figure 146](#).



**Figure 146. DMA\_MUX channel triggering: Ignored trigger**

This triggering capability may be used with any peripheral that supports eDMA transfers, and is most useful for periodically polling external devices on a particular bus.

As an example, the transmit side of a DSPI is assigned to a eDMA channel with a trigger, as described above. Once set up, the SPI will request eDMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the DSPI transfers can be automatically performed every 5  $\mu$ s (as an example). On the receive side of the SPI, the SPI and eDMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.

A more detailed description of the capability of each trigger (such as resolution, or range of values) may be found in the periodic interrupt timer chapter of the reference manual.

### 17.7.2 eDMA channels with no triggering capability

Channels 4–15 of the DMA\_MUX provide the normal routing functionality as described in [Section 17.3: Modes of operation](#).



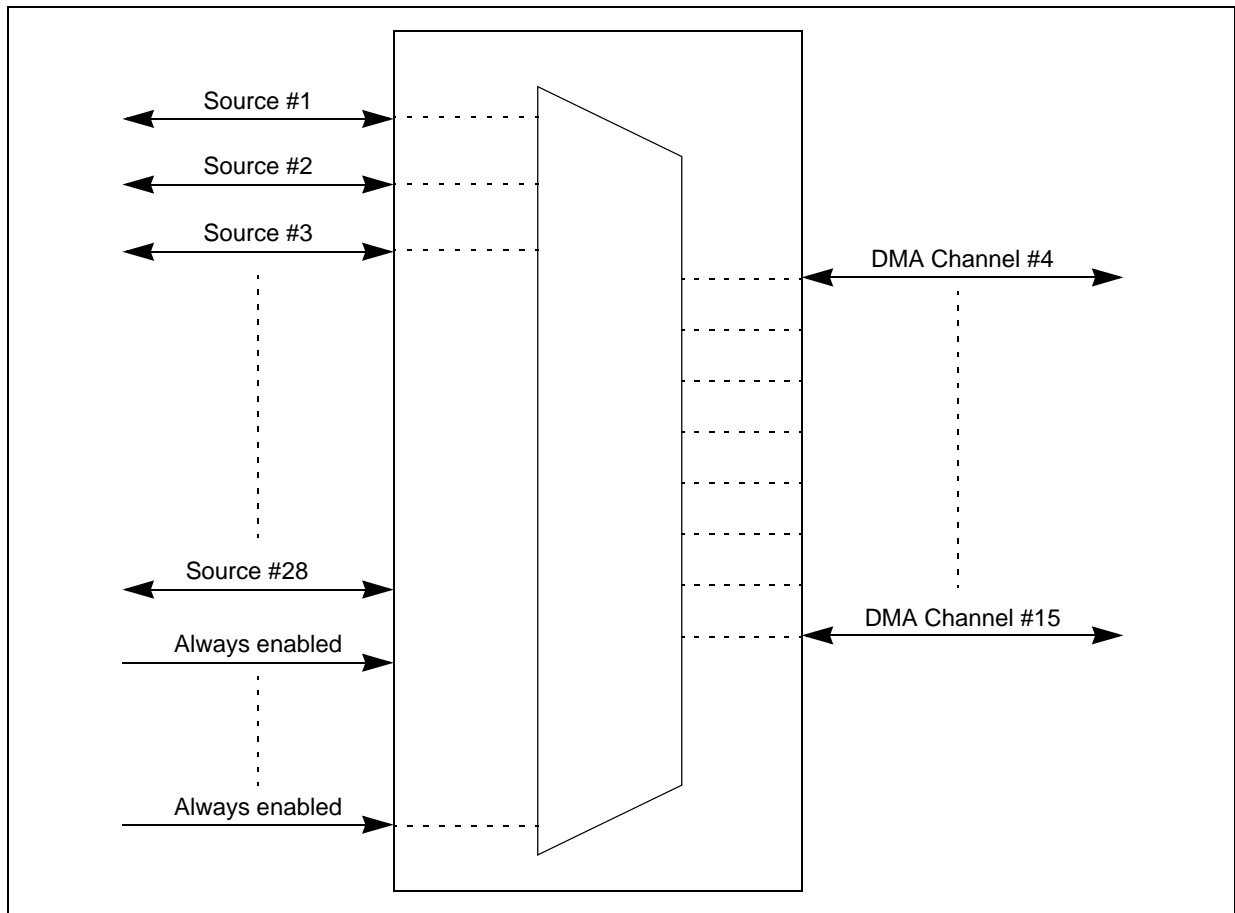


Figure 147. DMA\_MUX channel 4–15 block diagram

## 17.8 Initialization/Application information

### 17.8.1 Reset

The reset state of each individual bit is shown in [Section 17.5: Memory map and register definition](#). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

### 17.8.2 Enabling and configuring sources

#### 17.8.2.1 Enabling a source with periodic triggering

The following describes how to enable a source with periodic triggering:

1. Determine with which eDMA channel the source will be associated. Remember that only the first four eDMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the eDMA channel.
3. Ensure that the eDMA channel is properly configured in the eDMA. The eDMA channel may be enabled at this point.
4. In the PIT, configure the corresponding timer.
5. Select the source to be routed to the eDMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

**Example 1** Configure source #3 Transmit for use with eDMA Channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the eDMA, including enabling the channel
3. Configure Timer 4 in the Periodic Interrupt Timer (PIT) for the desired trigger interval
4. Write 0xC3 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #4 above:

```
In File registers.h:
    #define DMAMUX_BASE_ADDR      0xFC084000 /* Example only ! */
    /* Following example assumes char is 8-bits */
    volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
    (DMAMUX_BASE_ADDR+0x0002);

In File main.c:
    #include "registers.h"
    :
    :
    *CHCONFIG2 = 0x00;
    *CHCONFIG2 = 0xC3;
```

### 17.8.2.2 Enabling a source without periodic triggering

The following describes how to enable a source without periodic triggering:

1. Determine with which eDMA channel the source will be associated. Remember that only eDMA channels 0–3 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the eDMA channel.
3. Ensure that the eDMA channel is properly configured in the eDMA. The eDMA channel may be enabled at this point.
4. Select the source to be routed to the eDMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared.

**Example 2** Configure source #5 Transmit for use with eDMA Channel 2, without periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the eDMA, including enabling the channel
3. Write 0x85 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #3 above:

```
In File registers.h:
    #define DMAMUX_BASE_ADDR      0xFC084000 /* Example only ! */
    /* Following example assumes char is 8-bits */
```

```
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
```

```
In File main.c:
#include "registers.h"
:
:
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x85;
```

### 17.8.2.3 Disabling a source

A particular eDMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Additionally, some module specific configuration may be necessary. Please refer to the appropriate section for more details.

### 17.8.2.4 Switching the source of a eDMA channel

The following describes how to switch the source of a eDMA channel:

1. Disable the eDMA channel in the eDMA and reconfigure the channel for the new source.
2. Clear the ENBL and TRIG bits of the eDMA channel.
3. Select the source to be routed to the eDMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

#### **Example 3** Switch eDMA Channel 8 from source #5 transmit to source #7 transmit

1. In the eDMA configuration registers, disable eDMA channel 8 and re-configure it to handle the transfers to peripheral slot 7. This example assumes channel 8 doesn't have triggering capability.
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08)
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08).

The following code example illustrates steps #2 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR      0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);

In File main.c:
#include "registers.h"
:
:
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

## 18 Interrupt Controller (INTC)

### 18.1 Introduction

The INTC provides priority-based preemptive scheduling of interrupt service requests (ISRs). This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 204 interrupt requests. It is targeted to work with a Power Architecture technology processor and automotive powertrain applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.

For high priority interrupt requests in these target applications, the time from the assertion of the peripheral's interrupt request from the peripheral to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC supports this goal by providing a unique vector for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Since each individual application will have different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource cannot preempt each other.

Multiple processors can assert interrupt requests to each other through software configurable interrupt requests. These same software configurable interrupt requests also can be used to break the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software configurable interrupt request to finish the servicing in a lower priority ISR. Therefore these software configurable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

### 18.2 Features

- Supports 196 peripheral and 8 software-configurable interrupt request sources
- Unique 9-bit vector per interrupt source
- Each interrupt source can be programmed to one of 16 priorities
- Preemption
  - Preemptive prioritized interrupt requests to processor
  - ISR at a higher priority preempts ISRs or tasks at lower priorities
  - Automatic pushing or popping of preempted priority to or from a LIFO
  - Ability to modify the ISR or task priority; modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency – 3 clocks from receipt of interrupt request from peripheral to interrupt request to processor

Table 146. Interrupt sources available

Interrupt sources (204)	Number available
Software	8
ECSM	1
eDMA	17
Software Watchdog (SWT)	1
STM	4
Flash/SRAM ECC (SEC-DED)	2
Real Time Counter (RTC/API)	2
System Integration Unit Lite (SIUL)	3
WKPU	4
MC_ME	4
MC_RGM	1
FXOSC	1
SXOSC	1
PIT	8
ADC_0	2
ADC_1	2
FlexCAN_0	8
FlexCAN_1	8
FlexCAN_2	8
FlexCAN_3	8
FlexCAN_4	8
FlexCAN_5	8
LINFlex_0	3
LINFlex_1	3
LINFlex_2	3
LINFlex_3	3
LINFlex_4	3
LINFlex_5	3
LINFlex_6	3
LINFlex_7	3
LINFlex_8	3
LINFlex_9	3
DSPI_0	5
DSPI_1	5
DSPI_2	5

Table 146. Interrupt sources available (continued)

Interrupt sources (204)	Number available
DSPI_3	5
DSPI_4	5
DSPI_5	5
I2C_0	1
Enhanced Modular I/O Subsystem 0 (eMIOS_0)	16
eMIOS_1	16

### 18.3 Block diagram

Figure 148 provides a block diagram of the INTC.

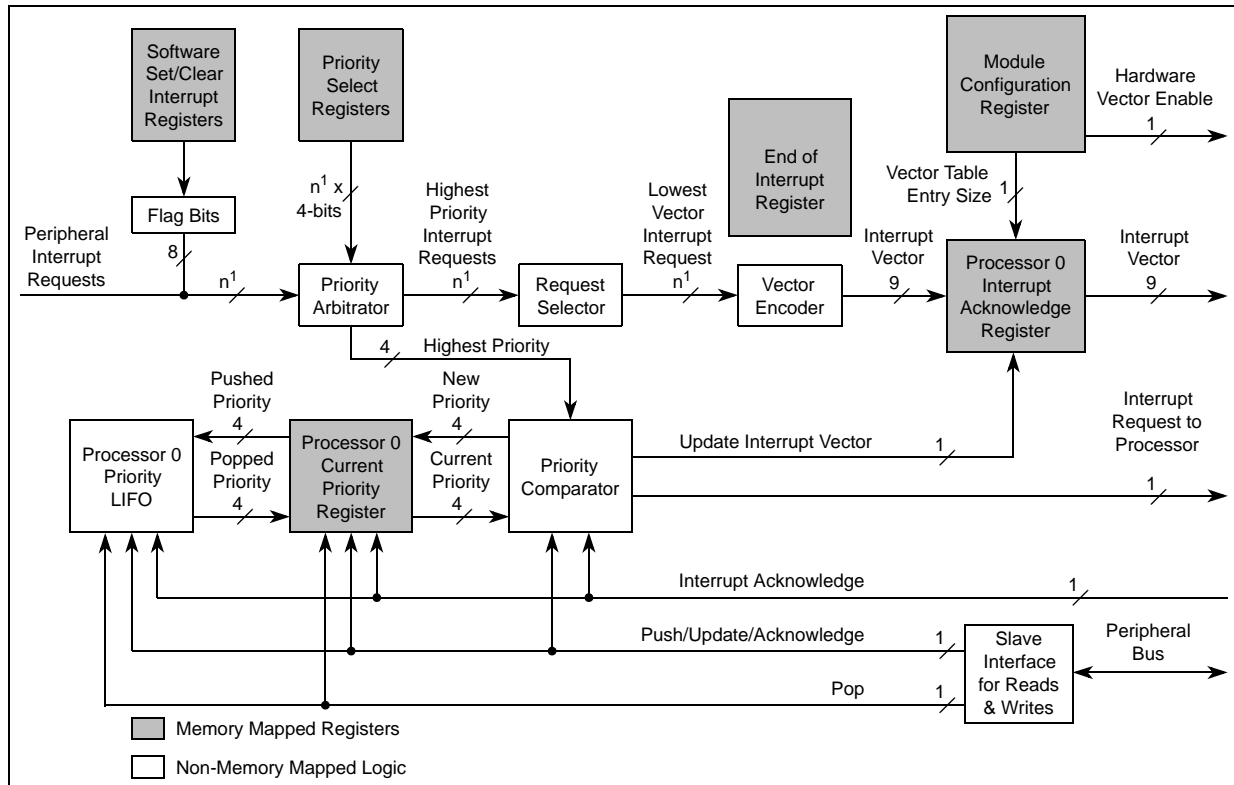


Figure 148. INTC block diagram

### 18.4 Modes of operation

#### 18.4.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

### 18.4.1.1 Software vector mode

In software vector mode, software, that is the interrupt exception handler, must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC will use software vector mode for a given processor when its associated HVEN bit in INTC\_MCR is negated. The hardware vector enable signal to processor 0 or processor 1 is driven as negated when its associated HVEN bit is negated. The vector is read from INC\_IACKR. Reading the INTC\_IACKR negates the interrupt request to the associated processor. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock. The reading also pushes the PRI value in INTC\_CPR onto the associated LIFO and updates PRI in the associated INTC\_CPR with the new priority.

Furthermore, the interrupt vector to the processor is driven as all 0s. The interrupt acknowledge signal from the associated processor is ignored.

### 18.4.1.2 Hardware vector mode

In hardware vector mode, the hardware is the interrupt vector signal from the INTC in conjunction with a processor with the capability use that vector. In hardware vector mode, this hardware causes the first instruction to be executed in handling the interrupt request to the processor to be specific to that vector. Therefore the interrupt exception handler is specific to a peripheral or software configurable interrupt request rather than being common to all of them. The INTC uses hardware vector mode for a given processor when the associated HVEN bit in the INTC\_MCR is asserted. The hardware vector enable signal to the associated processor is driven as asserted. When the interrupt request to the associated processor asserts, the interrupt vector signal is updated. The value of that interrupt vector is the unique vector associated with the preempting peripheral or software configurable interrupt request. The vector value matches the value of the INTVEC field in the INTC\_IACKR field in the INTC\_IACKR, depending on which processor was assigned to handle a given interrupt source.

The processor negates the interrupt request to the processor driven by the INTC by asserting the interrupt acknowledge signal for one clock. Even if a higher priority interrupt request arrived while waiting for the interrupt acknowledge, the interrupt request to the processor will negate for at least one clock.

The assertion of the interrupt acknowledge signal for a given processor pushes the associated PRI value in the associated INTC\_CPR register onto the associated LIFO and updates the associated PRI in the associated INTC\_CPR register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC\_CPR does not occur when the associated interrupt acknowledge signal asserts and INTC\_SSCIR0\_3–INTC\_SSCIR4\_7 is written at a time such that the PRI value in the associated INTC\_CPR register would need to be pushed and the previously last pushed PRI value would need to be popped simultaneously. In this case, PRI in the associated INTC\_CPR is updated with the new priority, and the associated LIFO is neither pushed or popped.

### 18.4.1.3 Debug mode

The INTC operation in debug mode is identical to its operation in normal mode.

**18.4.1.4 Stop mode**

The INTC supports STOP mode. The INTC can have its clock input disabled at any time by the clock driver on the device. While its clocks are disabled, the INTC registers are not accessible.

The INTC requires clocking in order for a peripheral interrupt request to generate an interrupt request to the processor. Since the INTC is not clocked in STOP mode, peripheral interrupt requests can not be used as a wakeup source, unless the device supports that interrupt request as a wakeup source.

**18.5 Memory map and register description**

**18.5.1 Module memory map**

*Table 147* shows the INTC memory map.

**Table 147. INTC memory map**

Base address: 0xFFF4_8000		
Address offset	Register	Location
0x0000	INTC Module Configuration Register (INTC_MCR)	<a href="#">18.5.2.1 on page 361</a>
0x0004	Reserved	
0x0008	INTC Current Priority Register for Processor (INTC_CPR)	<a href="#">18.5.2.2 on page 362</a>
0x000C	Reserved	
0x0010	INTC Interrupt Acknowledge Register (INTC_IACKR)	<a href="#">18.5.2.3 on page 363</a>
0x0014	Reserved	
0x0018	INTC End-of-Interrupt Register (INTC_EOIR)	<a href="#">18.5.2.4 on page 364</a>
0x001C	Reserved	
0x0020–0x0027	INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)	<a href="#">18.5.2.5 on page 365</a>
0x0028–0x003C	Reserved	
0x0040–0x00D0	INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR232_233) <sup>(1)</sup>	<a href="#">18.5.2.6 on page 366</a>

1. The PRI fields are “reserved” for peripheral interrupt requests whose vectors are labeled ‘Reserved’ in [Figure 150](#).

**18.5.2 Register description**

With exception of the INTC\_SSCIn and INTC\_PSRn, all registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of eight bits, aligned 16 bits, misaligned 16 bits to the middle two bytes, and aligned 32 bits.





Although INTC\_SSCIRn and INTC\_PSRn are 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of INTC\_IACKR are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC\_SSCIR0\_3–INTC\_SSCIR4\_7 or INTC\_EOIR does not affect the operation of the write.

**18.5.2.1 INTC Module Configuration Register (INTC\_MCR)**

The module configuration register is used to configure options of the INTC.

Offset: 0x0000 Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	VTES	0	0	0	0	0
W																HVEN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 149. INTC Module Configuration Register (INTC\_MCR)**

**Table 148. INTC\_MCR field descriptions**

Field	Description
VTES	Vector table entry size. Controls the number of '0's to the right of INTVEC in <a href="#">Section 18.5.2.3: INTC Interrupt Acknowledge Register (INTC_IACKR)</a> . If the contents of INTC_IACKR are used as an address of an entry in a vectortable as in software vector mode, then the number of rightmost '0's will determine the size of each vector table entry. VTES impacts software vector mode operation but also affects INTC_IACKR[INTVEC] position in both hardware vector mode and software vector mode. 0 4 bytes 1 8 bytes
HVEN	Hardware vector enable. Controls whether the INTC is in hardware vector mode or software vector mode. Refer to <a href="#">Section 18.4: Modes of operation</a> , for the details of the handshaking with the processor in each mode. 0 Software vector mode 1 Hardware vector mode

18.5.2.2 INTC Current Priority Register for Processor (INTC\_CPR)

Offset: 0x0008

Access: read/write

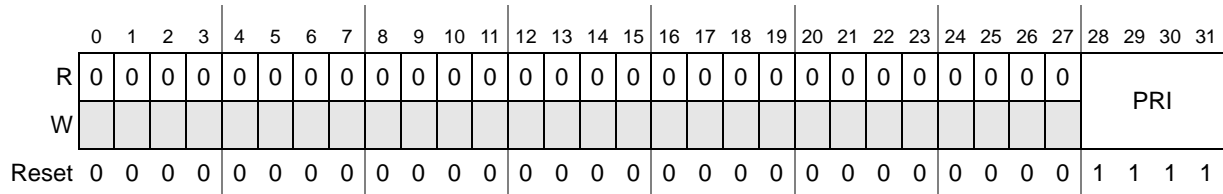


Figure 150. INTC Current Priority Register (INTC\_CPR)

Table 149. INTC\_CPR field descriptions

Field	Description
PRI	Priority PRI is the priority of the currently executing ISR according to the field values defined in <a href="#">Table 150</a> .

The INTC\_CPR masks any peripheral or software configurable interrupt request set at the same or lower priority as the current value of the INTC\_CPR[PRI] field from generating an interrupt request to the processor. When the INTC interrupt acknowledge register (INTC\_IACKR) is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When the INTC end-of-interrupt register (INTC\_EOIR) is written, the LIFO is popped into the INTC\_CPR's PRI field.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to [Section 18.7.5: Priority ceiling protocol](#).

*Note:* A store to modify the PRI field which closely precedes or follows an access to a shared resource can result in a non-coherent access to that resource. Refer to [Section 18.7.5.2: Ensuring coherency](#) for example code to ensure coherency.

Table 150. PRI values

PRI	Meaning
1111	Priority 15—highest priority
1110	Priority 14
1101	Priority 13
1100	Priority 12
1011	Priority 11
1010	Priority 10
1001	Priority 9
1000	Priority 8
0111	Priority 7
0110	Priority 6
0101	Priority 5

Table 150. PRI values (continued)

PRI	Meaning
0100	Priority 4
0011	Priority 3
0010	Priority 2
0001	Priority 1
0000	Priority 0—lowest priority

18.5.2.3 INTC Interrupt Acknowledge Register (INTC\_IACKR)

Offset: 0x0010 Access: read/write

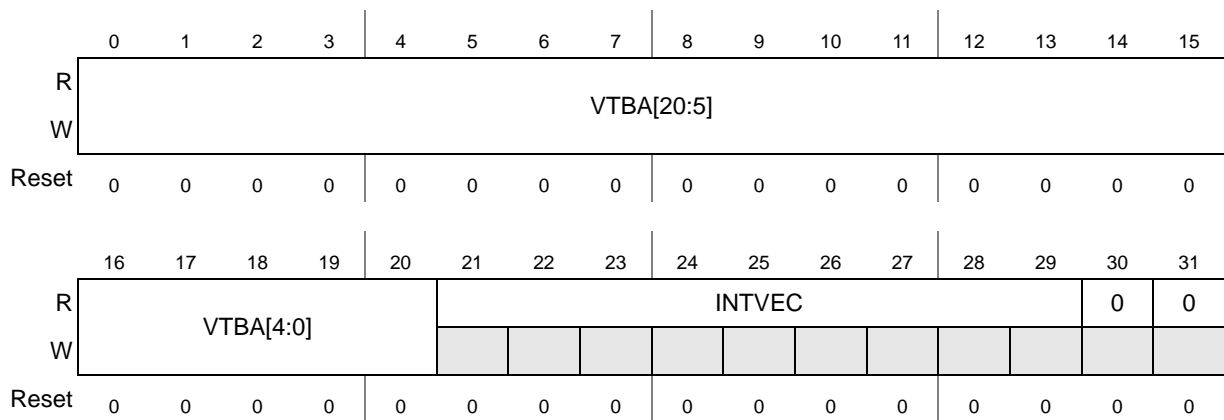


Figure 151. INTC Interrupt Acknowledge Register (INTC\_IACKR) when INTC\_MCR[VTES] = 0

Offset: 0x0010 Access: read/write

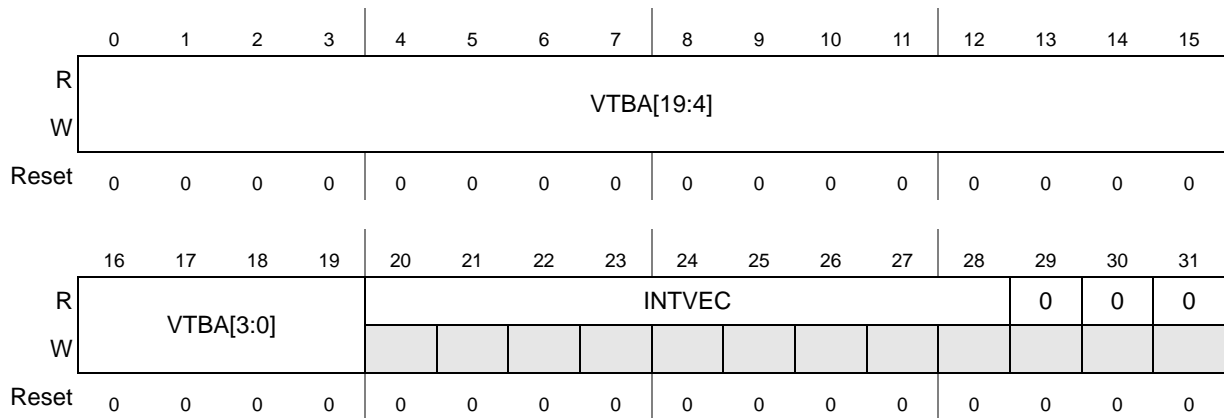


Figure 152. INTC Interrupt Acknowledge Register (INTC\_IACKR) when INTC\_MCR[VTES] = 1

**Table 151. INTC\_IACKR field descriptions**

Field	Description
VTBA	Vector Table Base Address Can be the base address of a vector table of addresses of ISRs.
INTVEC	Interrupt Vector It is the vector of the peripheral or software configurable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC is updated, whether the INTC is in software or hardware vector mode.

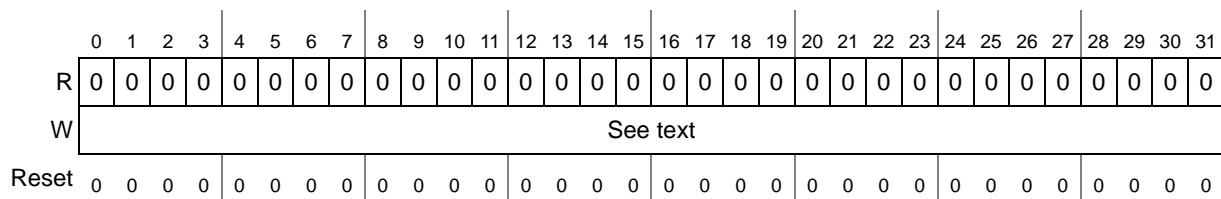
The interrupt acknowledge register provides a value which can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, the INTC\_IACKR has side effects from reads. Therefore, it must not be speculatively read while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC\_IACKR does not have side effects in hardware vector mode.

**18.5.2.4 INTC End-of-Interrupt Register (INTC\_EOIR)**

Offset: 0x0018

Access: Write only



**Figure 153. INTC End-of-Interrupt Register (INTC\_EOIR)**

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC\_EOIR is written, the priority last pushed on the LIFO is popped into INTC\_CPR. An exception to this behavior is described in [Section 18.4.1.2: Hardware vector mode](#). The values and size of data written to the INTC\_EOIR are ignored. The values and sizes written to this register neither update the INTC\_EOIR contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC\_EOIR.

Reading the INTC\_EOIR has no effect on the LIFO.

**18.5.2.5 INTC Software Set/Clear Interrupt Registers (INTC\_SSCIR0\_3–INTC\_SSCIR4\_7)**

Offset: 0x0020

Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR0	0	0	0	0	0	0	0	CLR1
W							SET0								SET1	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR2	0	0	0	0	0	0	0	CLR3
W							SET2								SET3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 154. INTC Software Set/Clear Interrupt Register 0–3 (INTC\_SSCIR[0:3])**

Offset: 0x0024

Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR4	0	0	0	0	0	0	0	CLR5
W							SET4								SET5	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR6	0	0	0	0	0	0	0	CLR7
W							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 155. INTC Software Set/Clear Interrupt Register 4–7 (INTC\_SSCIR[4:7])**

**Table 152. INTC\_SSCIR[0:7] field descriptions**

Field	Description
SETx	Set Flag Bits Writing a 1 sets the corresponding CLR <sub>x</sub> bit. Writing a 0 has no effect. Each SET <sub>x</sub> always will be read as a 0.
CLR <sub>x</sub>	Clear Flag Bits CLR <sub>x</sub> is the flag bit. Writing a 1 to CLR <sub>x</sub> clears it provided that a 1 is not written simultaneously to its corresponding SET <sub>x</sub> bit. Writing a 0 to CLR <sub>x</sub> has no effect. 0 Interrupt request not pending within INTC 1 Interrupt request pending within INTC

The software set/clear interrupt registers support the setting or clearing of software configurable interrupt request. These registers contain eight independent sets of bits to set

and clear a corresponding flag bit by software. Excepting being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC like a peripheral interrupt request. Writing a 1 to SETx will leave SETx unchanged at 0 but sets CLRx. Writing a 0 to SETx has no effect. CLRx is the flag bit. Writing a 1 to CLRx clears it. Writing a 0 to CLRx has no effect. If a 1 is written simultaneously to a pair of SETx and CLRx bits, CLRx will be asserted, regardless of whether CLRx was asserted before the write.

**18.5.2.6 INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233)**

Offset: 0x0040

Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI0				0	0	0	0	PRI1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI2				0	0	0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 156. INTC Priority Select Register 0–3 (INTC\_PSR[0:3])**

Offset: 0x0128

Access: read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI232				0	0	0	0	PRI233			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 157. INTC Priority Select Register 232-233 (INTC\_PSR[232:233])**

**Table 153. INTC\_PSR0\_3–INTC\_PSR232\_233 field descriptions**

Field	Description
PRI	Priority Select PRIx selects the priority for interrupt requests. See <a href="#">Section 18.6: Functional description</a> .

**Table 154. INTC Priority Select Register address offsets**

INTC_PSRx_x	Offset address	INTC_PSRx_x	Offset address
INTC_PSR0_3	0x0040	INTC_PSR120_123	0x00B8
INTC_PSR4_7	0x0044	INTC_PSR124_127	0x00BC
INTC_PSR8_11	0x0048	INTC_PSR128_131	0x00C0
INTC_PSR12_15	0x004C	INTC_PSR132_135	0x00C4
INTC_PSR16_19	0x0050	INTC_PSR136_139	0x00C8
INTC_PSR20_23	0x0054	INTC_PSR140_143	0x00CC
INTC_PSR24_27	0x0058	INTC_PSR144_147	0x00D0
INTC_PSR28_31	0x005C	INTC_PSR148_151	0x00D4
INTC_PSR32_35	0x0060	INTC_PSR152_155	0x00D8
INTC_PSR36_39	0x0064	INTC_PSR156_159	0x00DC
INTC_PSR40_43	0x0068	INTC_PSR160_163	0x00E0
INTC_PSR44_47	0x006C	INTC_PSR164_167	0x00E4
INTC_PSR48_51	0x0070	INTC_PSR168_171	0x00E8
INTC_PSR52_55	0x0074	INTC_PSR172_175	0x00EC
INTC_PSR56_59	0x0078	INTC_PSR176_179	0x00F0
INTC_PSR60_63	0x007C	INTC_PSR180_183	0x00F4
INTC_PSR64_67	0x0080	INTC_PSR184_187	0x00F8
INTC_PSR68_71	0x0084	INTC_PSR188_191	0x00FC
INTC_PSR72_75	0x0088	INTC_PSR192_195	0x0100
INTC_PSR76_79	0x008C	INTC_PSR196_199	0x0104
INTC_PSR80_83	0x0090	INTC_PSR200_203	0x0108
INTC_PSR84_87	0x0094	INTC_PSR204_207	0x010C
INTC_PSR88_91	0x0098	INTC_PSR208_211	0x0110
INTC_PSR92_95	0x009C	INTC_PSR212_215	0x0114
INTC_PSR96_99	0x00A0	INTC_PSR216_219	0x0118
INTC_PSR100_103	0x00A4	INTC_PSR220_223	0x011C
INTC_PSR104_107	0x00A8	INTC_PSR224_227	0x0120
INTC_PSR108_111	0x00AC	INTC_PSR228_231	0x0124
INTC_PSR112_115	0x00B0	INTC_PSR232_233	0x0128
INTC_PSR116_119	0x00B4		

## 18.6 Functional description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor.

*Note: The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request, whose PRIn value in INTC\_PSR0–INTC\_PSR233 is higher than the PRI value in INTC\_CPR, negates before the interrupt request to the processor for that peripheral or software settable interrupt request is acknowledged, the interrupt request to the processor still can assert or will remain asserted for that peripheral or software settable interrupt request. In this case, the interrupt vector will correspond to that peripheral or software settable interrupt request. Also, the PRI value in the INTC\_CPR will be updated with the corresponding PRIn value in INTC\_PSRn. Furthermore, clearing the peripheral interrupt request’s enable bit in the peripheral or, alternatively, setting its mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.*

**Table 155. Interrupt vector table**

IRQ #	Offset	Size (bytes)	Interrupt	Module
Section A (Core Section)				
—	0x0000	16	Critical Input (INTC software vector mode) / NMI	Core
—	0x0010	16	Machine check / NMI	Core
—	0x0020	16	Data Storage	Core
—	0x0030	16	Instruction Storage	Core
—	0x0040	16	External Input (INTC software vector mode)	Core
—	0x0050	16	Alignment	Core
—	0x0060	16	Program	Core
—	0x0070	16	Reserved	Core
—	0x0080	16	System call	Core
—	0x0090	96	Unused	Core
—	0x00F0	16	Debug	Core
—	0x0100	1792	Unused	Core
Section B (On-Platform Peripherals)				
0	0x0800	4	Software configurable flag 0	Software
1	0x0804	4	Software configurable flag 1	Software
2	0x0808	4	Software configurable flag 2	Software
3	0x080C	4	Software configurable flag 3	Software
4	0x0810	4	Software configurable flag 4	Software
5	0x0814	4	Software configurable flag 5	Software



Table 155. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
6	0x0818	4	Software configurable flag 6	Software
7	0x081C	4	Software configurable flag 7	Software
8	0x0820	4	Reserved	
9	0x0824	4	Platform Flash Bank 0 Abort   Platform Flash Bank 0 Stall   Platform Flash Bank 1 Abort   Platform Flash Bank 1 Stall	ECSM
10	0x0828	4	Combined Error	eDMA
11	0x082C	4	Channel 0	eDMA
12	0x0830	4	Channel 1	eDMA
13	0x0834	4	Channel 2	eDMA
14	0x0838	4	Channel 3	eDMA
15	0x083C	4	Channel 4	eDMA
16	0x0840	4	Channel 5	eDMA
17	0x0844	4	Channel 6	eDMA
18	0x0848	4	Channel 7	eDMA
19	0x084C	4	Channel 8	eDMA
20	0x0850	4	Channel 9	eDMA
21	0x0854	4	Channel 10	eDMA
22	0x0858	4	Channel 11	eDMA
23	0x085C	4	Channel 12	eDMA
24	0x0860	4	Channel 13	eDMA
25	0x0864	4	Channel 14	eDMA
26	0x0868	4	Channel 15	eDMA
27	0x086C	4	Reserved	
28	0x0870	4	Timeout	SWT
29	0x0874	4	Reserved	
30	0x0878	4	Match on channel 0	STM
31	0x087C	4	Match on channel 1	STM
32	0x0880	4	Match on channel 2	STM
33	0x0884	4	Match on channel 3	STM
34	0x0888	4	Reserved	
35	0x088C	4	ECC_DBD_PlatformFlash   ECC_DBD_PlatformRAM	Platform ECC Double Bit Detection

Table 155. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
36	0x0890	4	ECC_SBC_PlatformFlash   ECC_SBC_PlatformRAM	Platform ECC Single Bit Correction
37	0x0894	4	Reserved	
Section C				
38	0x0898	4	RTC	RTC/API
39	0x089C	4	API	RTC/API
40	0x08A0	4	Reserved	
41	0x08A4	4	SIU External IRQ_0	SIUL
42	0x08A8	4	SIU External IRQ_1	SIUL
43	0x08AC	4	SIU External IRQ_2	SIUL
44	0x08B0	4	Reserved	
45	0x08B4	4	Reserved	
46	0x08B8	4	WakeUp_IRQ_0	WKPU
47	0x08BC	4	WakeUp_IRQ_1	WKPU
48	0x08C0	4	WakeUp_IRQ_2	WKPU
49	0x08C4	4	WakeUp_IRQ_3	WKPU
50	0x08C8	4	Reserved	
51	0x08CC	4	Safe Mode Interrupt	MC_ME
52	0x08D0	4	Mode Transition Interrupt	MC_ME
53	0x08D4	4	Invalid Mode Interrupt	MC_ME
54	0x08D8	4	Invalid Mode Config	MC_ME
55	0x08DC	4	Reserved	
56	0x08E0	4	Functional and destructive reset alternate event interrupt (ipi_int)	MC_RGM
57	0x08E4	4	FXOSC counter expired (ipi_int_osc)	FXOSC
58	0x08E8	4	Reserved	
59	0x08EC	4	PITimer Channel 0	PIT
60	0x08F0	4	PITimer Channel 1	PIT
61	0x08F4	4	PITimer Channel 2	PIT
62	0x08F8	4	ADC_EOC	ADC_0
64	0x0900	4	ADC_WD	ADC_0
63	0x08FC	4	Reserved	
65	0x0904	4	FlexCAN_ESR[ERR_INT]	FlexCAN_0

Table 155. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
66	0x0908	4	FlexCAN_ESR_BOFF   FlexCAN_Transmit_Warning   FlexCAN_Receive_Warning	FlexCAN_0
67	0x090C	4	Reserved	
68	0x0910	4	FlexCAN_BUF_00_03	FlexCAN_0
69	0x0914	4	FlexCAN_BUF_04_07	FlexCAN_0
70	0x0918	4	FlexCAN_BUF_08_11	FlexCAN_0
71	0x091C	4	FlexCAN_BUF_12_15	FlexCAN_0
72	0x0920	4	FlexCAN_BUF_16_31	FlexCAN_0
73	0x0924	4	FlexCAN_BUF_32_63	FlexCAN_0
74	0x0928	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_0
75	0x092C	4	DSPI_SR[EOQF]	DSPI_0
76	0x0930	4	DSPI_SR[TFFF]	DSPI_0
77	0x0934	4	DSPI_SR[TCF]	DSPI_0
78	0x0938	4	DSPI_SR[RFDF]	DSPI_0
79	0x093C	4	LINFlex_RXI	LINFlex_0
80	0x0940	4	LINFlex_TXI	LINFlex_0
81	0x0944	4	LINFlex_ERR	LINFlex_0
82	0x0948	4	ADC_EOC	ADC_1
83	0x094C	4	Reserved	
84	0x0950	4	ADC_WD	ADC_1
85	0x0954	4	FlexCAN_ESR[ERR_INT]	FlexCAN_1
86	0x0958	4	FlexCAN_ESR_BOFF   FlexCAN_Transmit_Warning   FlexCAN_Receive_Warning	FlexCAN_1
87	0x095C	4	FlexCAN_ESR_WAK	FlexCAN_1
88	0x0960	4	FlexCAN_BUF_00_03	FlexCAN_1
89	0x0964	4	FlexCAN_BUF_04_07	FlexCAN_1
90	0x0968	4	FlexCAN_BUF_08_11	FlexCAN_1
91	0x096C	4	FlexCAN_BUF_12_15	FlexCAN_1
92	0x0970	4	FlexCAN_BUF_16_31	FlexCAN_1
93	0x0974	4	FlexCAN_BUF_32_63	FlexCAN_1
94	0x0978	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_1

Table 155. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
95	0x097C	4	DSPI_SR[EOQF]	DSPI_1
96	0x0980	4	DSPI_SR[TFFF]	DSPI_1
97	0x0984	4	DSPI_SR[TCF]	DSPI_1
98	0x0988	4	DSPI_SR[RFDF]	DSPI_1
99	0x098C	4	LINFlex_RXI	LINFlex_1
100	0x0990	4	LINFlex_TXI	LINFlex_1
101	0x0994	4	LINFlex_ERR	LINFlex_1
102	0x0998	4	Reserved	
103	0x099C	4	Reserved	
104	0x09A0	4	Reserved	
105	0x09A4	4	FlexCAN_[ERR_INT]	FlexCAN_2
106	0x09A8	4	FlexCAN_ESR_BOFF   FlexCAN_Transmit_Warning   FlexCAN_Receive_Warning	FlexCAN_2
107	0x09AC	4	Reserved	
108	0x09B0	4	FlexCAN_BUF_00_03	FlexCAN_2
109	0x09B4	4	FlexCAN_BUF_04_07	FlexCAN_2
110	0x09B8	4	FlexCAN_BUF_08_11	FlexCAN_2
111	0x09BC	4	FlexCAN_BUF_12_15	FlexCAN_2
112	0x09C0	4	FlexCAN_BUF_16_31	FlexCAN_2
113	0x09C4	4	FlexCAN_BUF_32_63	FlexCAN_2
114	0x09C8	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_2
115	0x09CC	4	DSPI_SR[EOQF]	DSPI_2
116	0x09D0	4	DSPI_SR[TFFF]	DSPI_2
117	0x09D4	4	DSPI_SR[TCF]	DSPI_2
118	0x09D8	4	DSPI_SR[RFDF]	DSPI_2
119	0x09DC	4	LINFlex_RXI	LINFlex_2
120	0x09E0	4	LINFlex_TXI	LINFlex_2
121	0x09E4	4	LINFlex_ERR	LINFlex_2
122	0x09E8	4	LINFlex_RXI	LINFlex_3
123	0x09EC	4	LINFlex_TXI	LINFlex_3
124	0x09F0	4	LINFlex_ERR	LINFlex_3

Table 155. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
125	0x09F4	4	I2C_SR[IBAL] I2C_SR[TCF] I2C_SR[IAAS]	I2C_0
126	0x09F8	4	Reserved	
127	0x09FC	4	PITimer Channel 3	PIT
128	0x0A00	4	PITimer Channel 4	PIT
129	0x0A04	4	PITimer Channel 5	PIT
130	0x0A08	4	PITimer Channel 6	PIT
131	0x0A0C	4	PITimer Channel 7	PIT
132	0x0A10	4	Reserved	
133	0x0A14	4	Reserved	
134	0x0A18	4	Reserved	
135	0x0A1C	4	Reserved	
136	0x0A20	4	Reserved	
137	0x0A24	4	Reserved	
138	0x0A28	4	Reserved	
139	0x0A2C	4	Reserved	
140	0x0A30	4	Reserved	
141	0x0A34	4	EMIOS_GFR[F0,F1]	eMIOS_0
142	0x0A38	4	EMIOS_GFR[F2,F3]	eMIOS_0
143	0x0A3C	4	EMIOS_GFR[F4,F5]	eMIOS_0
144	0x0A40	4	EMIOS_GFR[F6,F7]	eMIOS_0
145	0x0A44	4	EMIOS_GFR[F8,F9]	eMIOS_0
146	0x0A48	4	EMIOS_GFR[F10,F11]	eMIOS_0
147	0x0A4C	4	EMIOS_GFR[F12,F13]	eMIOS_0
148	0x0A50	4	EMIOS_GFR[F14,F15]	eMIOS_0
149	0x0A54	4	EMIOS_GFR[F16,F17]	eMIOS_0
150	0x0A58	4	EMIOS_GFR[F18,F19]	eMIOS_0
151	0x0A5C	4	EMIOS_GFR[F20,F21]	eMIOS_0
152	0x0A60	4	EMIOS_GFR[F22,F23]	eMIOS_0
153	0x0A64	4	EMIOS_GFR[F24,F25]	eMIOS_0
154	0x0A68	4	EMIOS_GFR[F26,F27]	eMIOS_0
155	0x0A6C	4	EMIOS_GFR[F28,F29]	eMIOS_0
156	0x0A70	4	EMIOS_GFR[F30,F31]	eMIOS_0
Section D (Device specific vectors)				

Table 155. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
157	0x0A74	4	EMIOS_GFR[F0,F1]	eMIOS_1
158	0x0A78	4	EMIOS_GFR[F2,F3]	eMIOS_1
159	0x0A7C	4	EMIOS_GFR[F4,F5]	eMIOS_1
160	0x0A80	4	EMIOS_GFR[F6,F7]	eMIOS_1
161	0x0A84	4	EMIOS_GFR[F8,F9]	eMIOS_1
162	0x0A88	4	EMIOS_GFR[F10,F11]	eMIOS_1
163	0x0A8C	4	EMIOS_GFR[F12,F13]	eMIOS_1
164	0x0A90	4	EMIOS_GFR[F14,F15]	eMIOS_1
165	0x0A94	4	EMIOS_GFR[F16,F17]	eMIOS_1
166	0x0A98	4	EMIOS_GFR[F18,F19]	eMIOS_1
167	0x0A9C	4	EMIOS_GFR[F20,F21]	eMIOS_1
168	0x0AA0	4	EMIOS_GFR[F22,F23]	eMIOS_1
169	0x0AA4	4	EMIOS_GFR[F24,F25]	eMIOS_1
170	0x0AA8	4	EMIOS_GFR[F26,F27]	eMIOS_1
171	0x0AAC	4	EMIOS_GFR[F28,F29]	eMIOS_1
172	0x0AB0	4	EMIOS_GFR[F30,F31]	eMIOS_1
173	0x0AB4	4	FlexCAN_ESR	FlexCAN_3
174	0x0AB8	4	FlexCAN_ESR_BOFF   FlexCAN_Transmit_Warning   FlexCAN_Receive_Warning	FlexCAN_3
175	0x0ABC	4	Reserved	
176	0x0AC0	4	FlexCAN_BUF_0_3	FlexCAN_3
177	0x0AC4	4	FlexCAN_BUF_4_7	FlexCAN_3
178	0x0AC8	4	FlexCAN_BUF_8_11	FlexCAN_3
179	0x0ACC	4	FlexCAN_BUF_12_15	FlexCAN_3
180	0x0AD0	4	FlexCAN_BUF_16_31	FlexCAN_3
181	0x0AD4	4	FlexCAN_BUF_32_63	FlexCAN_3
182	0x0AD8	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_3
183	0x0ADC	4	DSPI_SR[EOQF]	DSPI_3
184	0x0AE0	4	DSPI_SR[TFFF]	DSPI_3
185	0x0AE4	4	DSPI_SR[TCF]	DSPI_3
186	0x0AE8	4	DSPI_SR[RFDF]	DSPI_3
187	0x0AEC	4	LINFlex_RXI	LINFlex_4
188	0x0AF0	4	LINFlex_TXI	LINFlex_4

Table 155. Interrupt vector table (continued)

IRQ #	Offset	Size (bytes)	Interrupt	Module
189	0x0AF4	4	LINFlex_ERR	LINFlex_4
190	0x0AF8	4	FlexCAN_ESR	FlexCAN_4
191	0x0AFC	4	FlexCAN_ESR_BOFF   FlexCAN_Transmit_Warning   FlexCAN_Receive_Warning	FlexCAN_4
192	0x0B00	4	Reserved	
193	0x0B04	4	FlexCAN_BUF_0_3	FlexCAN_4
194	0x0B08	4	FlexCAN_BUF_4_7	FlexCAN_4
195	0x0B0C	4	FlexCAN_BUF_8_11	FlexCAN_4
196	0x0B10	4	FlexCAN_BUF_12_15	FlexCAN_4
197	0x0B14	4	FlexCAN_BUF_16_31	FlexCAN_4
198	0x0B18	4	FlexCAN_BUF_32_63	FlexCAN_4
199	0x0B1C	4	LINFlex_RXI	LINFlex_5
200	0x0B20	4	LINFlex_TXI	LINFlex_5
201	0x0B24	4	LINFlex_ERR	LINFlex_5
202	0x0B28	4	FlexCAN_ESR	FlexCAN_5
203	0x0B2C	4	FlexCAN_ESR_BOFF   FlexCAN_Transmit_Warning   FlexCAN_Receive_Warning	FlexCAN_5
204	0x0B30	4	Reserved	
205	0x0B34	4	FlexCAN_BUF_0_3	FlexCAN_5
206	0x0B38	4	FlexCAN_BUF_4_7	FlexCAN_5
207	0x0B3C	4	FlexCAN_BUF_8_11	FlexCAN_5
208	0x0B40	4	FlexCAN_BUF_12_15	FlexCAN_5
209	0x0B44	4	FlexCAN_BUF_16_31	FlexCAN_5
210	0x0B48	4	FlexCAN_BUF_32_63	FlexCAN_5
211	0x0B4C	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_4
212	0x0B50	4	DSPI_SR[EOQF]	DSPI_4
213	0x0B54	4	DSPI_SR[TFFF]	DSPI_4
214	0x0B58	4	DSPI_SR[TCF]	DSPI_4
215	0x0B5C	4	DSPI_SR[RFDF]	DSPI_4
216	0x0B60	4	LINFlex_RXI	LINFlex_6
217	0x0B64	4	LINFlex_TXI	LINFlex_6
218	0x0B68	4	LINFlex_ERR	LINFlex_6

**Table 155. Interrupt vector table (continued)**

IRQ #	Offset	Size (bytes)	Interrupt	Module
219	0x0B6C	4	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_5
220	0x0B70	4	DSPI_SR[EOQF]	DSPI_5
221	0x0B74	4	DSPI_SR[TFFF]	DSPI_5
222	0x0B78	4	DSPI_SR[TCF]	DSPI_5
223	0x0B7C	4	DSPI_SR[RFDF]	DSPI_5
224	0x0B80	4	LINFlex_RXI	LINFlex_7
225	0x0B84	4	LINFlex_TXI	LINFlex_7
226	0x0B88	4	LINFlex_ERR	LINFlex_7
227	0x0B8C	4	LINFlex_RXI	LINFlex_8
228	0x0B90	4	LINFlex_TXI	LINFlex_8
229	0x0B94	4	LINFlex_ERR	LINFlex_8
230	0x0B98	4	LINFlex_RXI	LINFlex_9
231	0x0B9C	4	LINFlex_TXI	LINFlex_9
232	0x0BA0	4	LINFlex_ERR	LINFlex_9
233	0x0BA4	4	32KXOSC counter expired	SXOSC

**18.6.1 Interrupt request sources**

The INTC has two types of interrupt requests, peripheral and software configurable. These interrupt requests can assert on any clock cycle.

**18.6.1.1 Peripheral interrupt requests**

An interrupt event in a peripheral’s hardware sets a flag bit that resides in the peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

External interrupts are handled by the SIU (see [Section 21.6.3: External interrupts](#)).

**18.6.1.2 Software configurable interrupt requests**

An interrupt request is triggered by software by writing a 1 to a SETx bit in *INTC\_SSCIR0\_3–INTC\_SSCIR4\_7*. This write sets the corresponding flag bit, CLR<sub>x</sub>, resulting in the interrupt request. The interrupt request is cleared by writing a 1 to the CLR<sub>x</sub> bit.

The time from the write to the SETx bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.



### 18.6.1.3 Unique vector for each interrupt request source

Each peripheral and software configurable interrupt request is assigned a hardwired unique 9-bit vector. Software configurable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests. The peripheral interrupt request input ports at the boundary of the INTC block are assigned specific hardwired vectors within the INTC (see [Table 146](#)).

## 18.6.2 Priority management

The asserted interrupt requests are compared to each other based on their PRIx values set in the INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233). The result is compared to PRI in the associated *INTC\_CPR*. The results of those comparisons manage the priority of the ISR executed by the associated processor. The associated LIFO also assists in managing that priority.

### 18.6.2.1 Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in [Figure 148](#) compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software configurable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software configurable interrupt request is generated for INTC interrupt acknowledge register (INTC\_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

#### 18.6.2.1.1 Priority arbitrator subblock

The priority arbitrator subblock for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software configurable. The output of the priority arbitrator subblock is the highest of those priorities assigned to a given processor. Also, any interrupt requests which have this highest priority are output as asserted interrupt requests to the associated request selector subblock.

#### 18.6.2.1.2 Request selector subblock

If only one interrupt request from the associated priority arbitrator subblock is asserted, then it is passed as asserted to the associated vector encoder subblock. If multiple interrupt requests from the associated priority arbitrator subblock are asserted, the only the one with the lowest vector is passed as asserted to the associated vector encoder subblock. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software configurable interrupt requests.

#### 18.6.2.1.3 Vector encoder subblock

The vector encoder subblock generates the unique 9-bit vector for the asserted interrupt request from the request selector subblock for the associated processor.

#### 18.6.2.1.4 Priority Comparator subblock

The priority comparator subblock compares the highest priority output from the priority arbitrator subblock with PRI in *INTC\_CPR*. If the priority comparator subblock detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the associated processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in *INTC\_CPR* or the PRI value in

INTC\_CPR is lowered below this highest priority. This highest priority then becomes the new priority which will be written to PRI in INTC\_CPR when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI $n$  in INTC\_PSR $n$  are zero will not cause a preemption because their PRI $n$  will not be higher than PRI in INTC\_CPR.

### 18.6.2.2 Last-In First-Out (LIFO)

The LIFO stores the preempted PRI values from the INTC\_CPR. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the INTC\_CPR does not need to be loaded from the INTC\_CPR and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the INTC\_CPR.

The PRI value in the INTC\_CPR is pushed onto the LIFO when the INTC\_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC\_CPR whenever the INTC\_EOIR is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC\_CPR equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop '0's if it is popped more times than it is pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

## 18.6.3 Handshaking with processor

### 18.6.3.1 Software vector mode handshaking

This section describes handshaking in software vector mode.

#### 18.6.3.1.1 Acknowledging interrupt request to processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in [Figure 158](#). The INTC examines the peripheral and software configurable interrupt requests. When it finds an asserted peripheral or software configurable interrupt request with a higher priority than PRI in the associated INTC\_CPR, it asserts the interrupt request to the processor. The INTVEC field in the associated INTC\_IACKR is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of the handshaking is described in [Section 18.4.1.1: Software vector mode](#).

#### 18.6.3.1.2 End of interrupt exception handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC\_EOIR) must be written. When written, the associated LIFO is popped so the preempted priority is restored into PRI of the INTC\_CPR. Before it is written, the peripheral

or software configurable flag bit must be cleared so that the peripheral or software configurable interrupt request is negated.

*Note:* To ensure proper operation across all Power Architecture® MCUs, execute an MBAR or MSYNC instruction between the access to clear the flag bit and the write to the INTC\_EOIR.

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer even be asserted. When PRI in INTC\_CPR is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software settable interrupt request at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor will return to the instruction address that it was to next execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

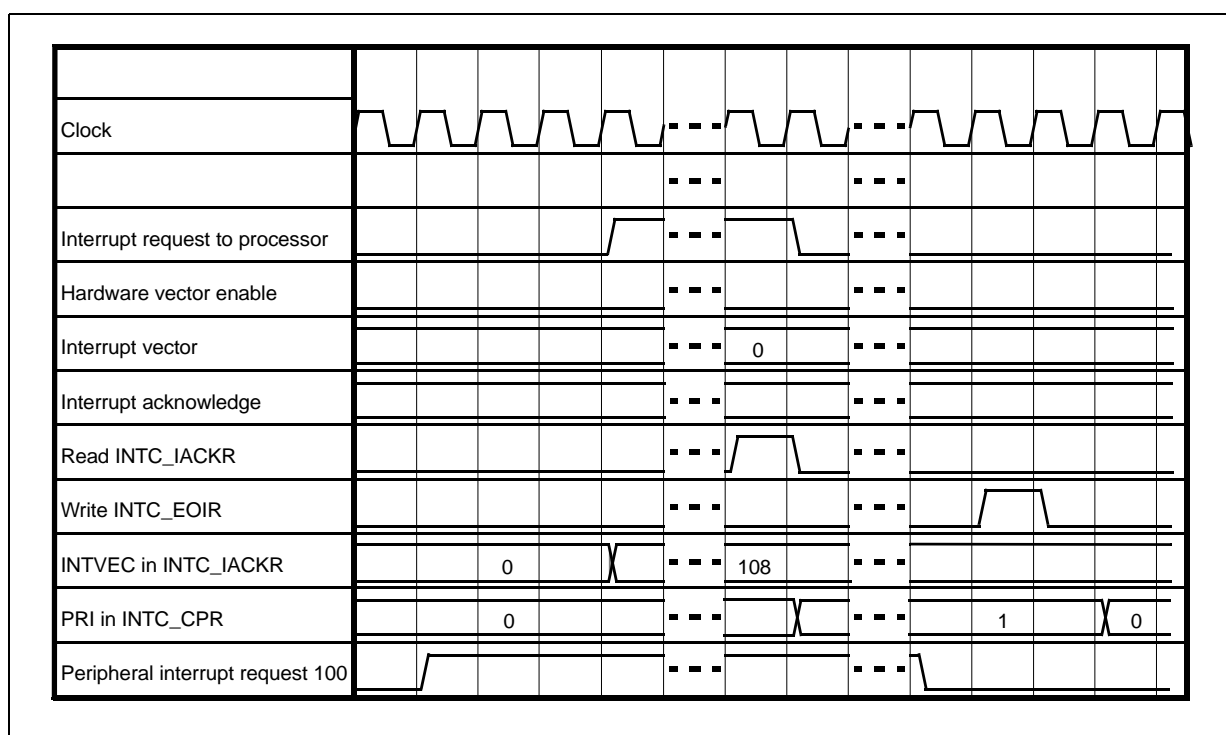


Figure 158. Software vector mode handshaking timing diagram

### 18.6.3.2 Hardware vector mode handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in [Figure 159](#). As in software vector mode, the INTC examines the peripheral and software settable interrupt requests, and when it finds an asserted one with a higher priority than PRI in INTC\_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC\_IACKR is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC

field in the INTC\_IACKR. The rest of the handshaking is described in [Section 18.7.2.2: Hardware vector mode](#).

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC\_EOIR, is the same as in software vector mode. Refer to [Section 18.6.3.1.2: End of interrupt exception handler](#).

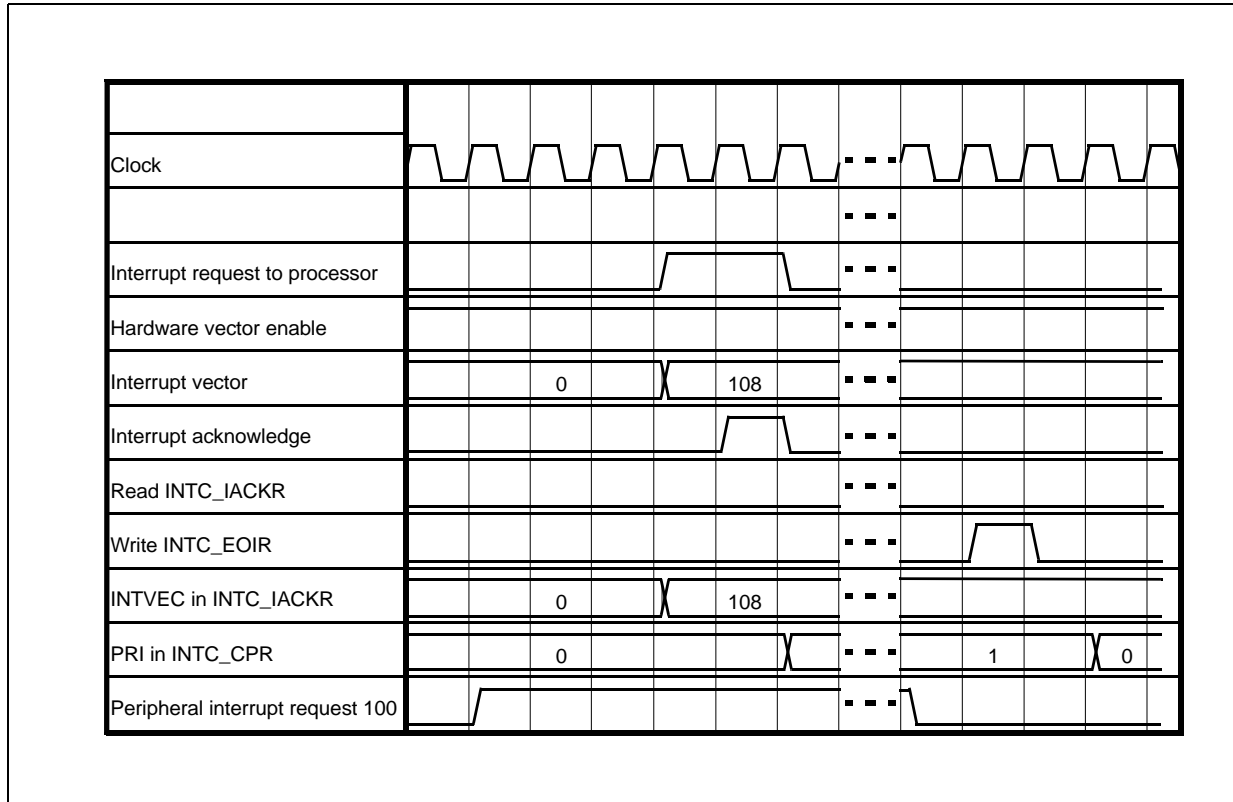


Figure 159. Hardware vector mode handshaking timing diagram

## 18.7 Initialization/application information

### 18.7.1 Initialization flow

After exiting reset, all of the  $PRI_n$  fields in INTC priority select registers (INTC\_PSR0–INTC\_PSR233) will be zero, and PRI in INTC current priority register (INTC\_CPR) will be 15. These reset values will prevent the INTC from asserting the interrupt request to the processor. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated. An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is:

```

interrupt_request_initialization:
    configure VTES and HVEN in INTC_MCR
    configure VTBA in INTC_IACKR
    raise the  $PRI_n$  fields in INTC_PSR $n$ 
    set the enable bits or clear the mask bits for the peripheral interrupt
    requests
    
```

```

lower PRI in INTC_CPR to zero
enable processor recognition of interrupts

```

## 18.7.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture™ assembly code.

### 18.7.2.1 Software vector mode

```

interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1
lis r3,INTC_IACKR@ha # form adjusted upper half of INTC_IACKR address
lwz r3,INTC_IACKR@l(r3) # load INTC_IACKR, which clears request to processor
lwz r3,0x0(r3) # load address of ISR from vector table
wrteei 1 # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr r3 # move INTC_IACKR contents into link register
blr1 # branch to ISR; link register updated with epilog
# address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
# disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
# at the cost of
# postponing the servicing of the next interrupt request.
mbar # ensure store to clear flag bit has completed
lis r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li r4,0x0 # form 0 to write to INTC_EOIR
wrteei 0 # disable processor recognition of interrupts
stw r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi

vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
.
.
.
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511

ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr # return to epilog

```

### 18.7.2.2 Hardware vector mode

This interrupt exception handler is useful with processor and system bus implementations which support a hardware vector. This example assumes that each `interrupt_exception_handlerx` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` is needed.

```

interrupt_exception_handlerx:
b interrupt_exception_handler_continuedx# 4 instructions available, branch
to continue
interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei 1 # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl ISRx # branch to ISR for interrupt with vector x

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar # ensure store to clear flag bit has completed
lis r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li r4,0x0 # form 0 to write to INTC_EOIR
wrteei 0 # disable processor recognition of interrupts
stw r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi

ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC
blr # branch to epilog

```

### 18.7.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC\_CPR) having a value of 0. The RTOS will execute the tasks according to whatever priority scheme that it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC\_CPR priority 0 and outside the control of the RTOS, the RTOS executes at INTC\_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC\_CPR priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC\_CPR while the shared resource is being accessed.

An ISR whose *PRI<sub>n</sub>* in INTC priority select registers (INTC\_PSR0–INTC\_PSR233) has a value of 0 will not cause an interrupt request to the processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit will cause it to remain negated, which consequently also will not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR will not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

### 18.7.4 Order of execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software configurable interrupt requests. However, if multiple peripheral or software configurable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software configurable interrupt requests asserted.

The example in [Table 156](#) shows the order of execution of both ISRs with different priorities and the same priority.

**Table 156. Order of ISR execution example**

Step No.	Step description	Code Executing at End of Step					Interrupt exception handler	PRI in INTC_CPR at End of Step
		RTOS	ISR108 (1)	ISR208	ISR308	ISR408		
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR.						X	1

Table 156. Order of ISR execution example (continued)

Step No.	Step description	Code Executing at End of Step					Interrupt exception handler	PRI in INTC_CPR at End of Step
		RTOS	ISR108 <sup>(1)</sup>	ISR208	ISR308	ISR408		
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR.						X	0
12	RTOS continues execution.	X						0

1. ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software configurable interrupt requests.

### 18.7.5 Priority ceiling protocol

#### 18.7.5.1 Elevating priority

The PRI field in *INTC\_CPR* is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in *INTC\_CPR* to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in *INTC\_CPR* can be lowered. If they do not raise their priority, ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

#### 18.7.5.2 Ensuring coherency

A scenario can cause non-coherent accesses to the shared resource. For example, ISR1 and ISR2 are both running on the same core and both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing and writes to the *INTC\_CPR*. The instruction following this store is a store to a value in a shared coherent data block. Either immediately before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible that both stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.





OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corruption of a coherent data block, modifications to PRI in INTC\_CPR can be made by those system services with the code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

### 18.7.6 Selecting priorities according to request rates and deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs which have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100  $\mu$ s, ISR2 executes every 200  $\mu$ s, and ISR3 executes every 300  $\mu$ s. ISR1 has a higher priority than ISR2 which has a higher priority than ISR3; however, if ISR3 has a deadline of 150  $\mu$ s, then it has a higher priority than ISR2.

The INTC has 16 priorities, which may be less than the number of ISRs. In this case, the ISRs should be grouped with other ISRs that have similar deadlines. For example, a priority could be allocated for every time the request rate doubles. ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500  $\mu$ s would share a priority, ISRs with request rates around 250  $\mu$ s would share a priority, etc. With this approach, a range of ISR request rates of  $2^{16}$  could be included, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor's ability to meet its deadlines. However, reducing the number of priorities can reduce the size and latency through the interrupt controller. It also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

### 18.7.7 Software configurable interrupt requests

The software configurable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and they may also be used by processors to interrupt other processors in a multiple processor system.

#### 18.7.7.1 Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the PRIx value in the INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233), which becomes the PRI value in INTC\_CPR with the interrupt acknowledge. The ISR, however, can have a portion that does not need to be executed at this higher priority. Therefore, executing the later portion that does not need to be executed at this higher priority can prevent the execution of ISRs which do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SETx bit in INTC\_SSCIR0\_3–INTC\_SSCIR4\_7. Writing a 1 to SETx causes a software configurable interrupt request. This software configurable interrupt request will usually have a lower PRIx

value in the INTC\_PSRx\_x and will not cause preemptive scheduling inefficiencies. After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

### 18.7.7.2 Scheduling an ISR on another processor

Because the SETx bits in the INTC\_SSCIRx\_x are memory mapped, processors in multiple-processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that the processor executing the software configurable ISR has not completed the work before asking it to again execute the ISR, it can check if the corresponding CLRx bit in INTC\_SSCIRx\_x is asserted before again writing a 1 to the SETx bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. To do this, the first processor writes a 1 to a SETx bit on the second processor. After accessing the block of data, the second processor clears the corresponding CLRx bit and then writes 1 to a SETx bit on the first processor, informing it that it can now access the block of data.

### 18.7.8 Lowering priority within an ISR

A common method for avoiding preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities (see [Section 18.7.7.1: Scheduling a lower priority portion of an ISR](#)) is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

*Note:* Lowering the PRI value in INTC\_CPR within an ISR to below the ISR's corresponding PRI value in the INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233) allows more preemptions than the LIFO depth can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

### 18.7.9 Negating an interrupt request outside of its ISR

#### 18.7.9.1 Negating an interrupt request as a side effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits and their corresponding interrupt requests. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

#### 18.7.9.2 Negating multiple interrupt requests in one ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

# 19 Crossbar Switch (XBAR)

## 19.1 Introduction

This chapter describes the multi-port crossbar switch (XBAR), which supports simultaneous connections between three master ports and three slave ports. XBAR supports a 32-bit address bus width and a 32-bit data bus width at all master and slave ports.

## 19.2 Block diagram

Figure 160 shows a block diagram of the crossbar switch.

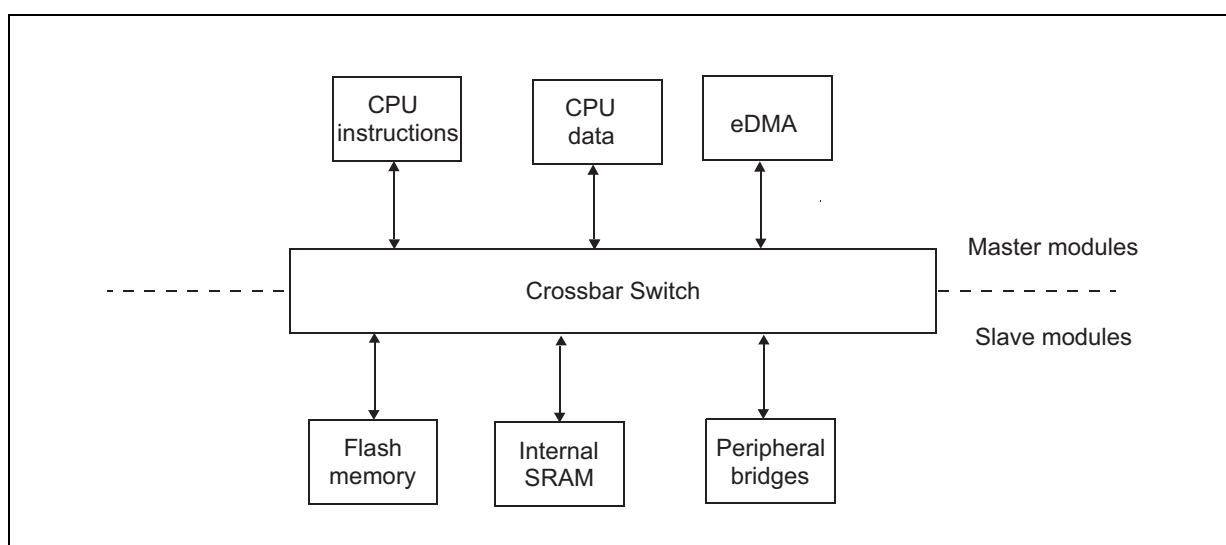


Figure 160. XBAR block diagram

Table 157 gives the crossbar switch port for each master and slave, and the assigned and fixed ID number for each master. The table shows the master ID numbers as they relate to the master port numbers.

Table 157. XBAR switch ports for SPC560B54/6x

Module	Port		Physical master ID
	Type	Logical number	
e200z0 core–CPU instructions	Master	0	0
e200z0 core–CPU data / Nexus	Master	0	1
Flash memory	Slave	0	—
Internal SRAM	Slave	2	—
Peripheral bridges	Slave	7	—
eDMA	Master	1	2

## 19.3 Overview

The XBAR allows for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

Requesting masters are granted access based on a fixed priority.

## 19.4 Features

- 3 master ports:
  - Core: e200z0 core instructions
  - Core: e200z0 core data / Nexus
- 3 slave ports
  - Flash (refer to the flash memory chapter for information on accessing flash memory)
  - Internal SRAM
  - Peripheral bridges
- 32-bit address, 32-bit data paths
- Fully concurrent transfers between independent master and slave ports
- Fixed priority scheme and fixed parking strategy
- eDMA

## 19.5 Modes of operation

### 19.5.1 Normal mode

In normal mode, the XBAR provides the logic that controls crossbar switch configuration.

### 19.5.2 Debug mode

The XBAR operation in debug mode is identical to operation in normal mode.

## 19.6 Functional description

This section describes the functionality of the XBAR in more detail.

### 19.6.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. To maximize data throughput, it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR stalls masters, or inserts bubbles on the slave side.

## 19.6.2 General operation

When a master makes an access to the XBAR from an idle master state, the access is taken immediately by the XBAR. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master's priority level and the responding slave's access time.

Because the XBAR appears to be simply another slave to the master device, the master device has no indication that it owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of a targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed.

When a slave bus is idled by the XBAR, it is parked on the master which did the last transfer.

## 19.6.3 Master ports

A master access is taken if the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case, the XBAR is completely transparent and the master access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stall if the access decodes to a slave port that is busy serving another master, parked on another master.

If the slave port is currently parked on another master, and no other master is requesting access to the slave port, then only one clock of arbitration is incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters, then the master gains control over the slave port as soon as the data phase of the current access is completed. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR directly responds with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

### 19.6.4 Slave ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. To do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR forces a bubble onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master which does not own the slave port is granted access after a one clock delay.

### 19.6.5 Priority assignment

Each master port is assigned a fixed 3-bit priority level (hard-wired priority). The following table shows the priority levels assigned to each master (the lowest has highest priority).

**Table 158. Hardwired bus master priorities**

Module	Port		Priority level
	Type	Number	
e200z0 core-CPU instructions	Master	0	7
e200z0 core-CPU data	Master	1	6
eDMA	Master	2	5

### 19.6.6 Arbitration

XBAR supports only a fixed-priority comparison algorithm.

#### 19.6.6.1 Fixed priority operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR\_MPR. If two masters both request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master’s priority level is higher than that of the master that currently has control over the slave port (if any). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master’s priority level is higher than that of the master that currently has control of the slave port, the higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But if the new requesting master’s priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port is finished accessing the current slave port.

**19.6.6.1.1 Parking**

If no master is currently requesting the slave port, the slave port is parked. The slave port parks always to the last master (park-on-last). When parked on the last master, the slave port is passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.

## 20 Memory Protection Unit (MPU)

### 20.1 Introduction

The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in the device. Using preprogrammed region descriptors which define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU module provides the following capabilities:

- Support for 8 program-visible 128-bit (4-word) region descriptors
  - Each region descriptor defines a modulo-32 byte space, aligned anywhere in memory  
Region sizes can vary from a minimum of 32 bytes to a maximum of 4 Gbytes
  - Two types of access control permissions defined in single descriptor word  
Processors have separate {read, write, execute} attributes for supervisor and user accesses  
Non-processor masters have {read, write} attributes
  - Hardware-assisted maintenance of the descriptor valid bit minimizes coherency issues
  - Alternate programming model view of the access control permissions word
- Memory-mapped platform device
  - Interface to 3 slave XBAR ports: flash controller, system SRAM controller and peripherals bus  
Connections to the address phase address and attributes  
Typical location is immediately “downstream” of the platform’s crossbar switch

A simplified block diagram of the MPU module is shown in [Figure 161](#).



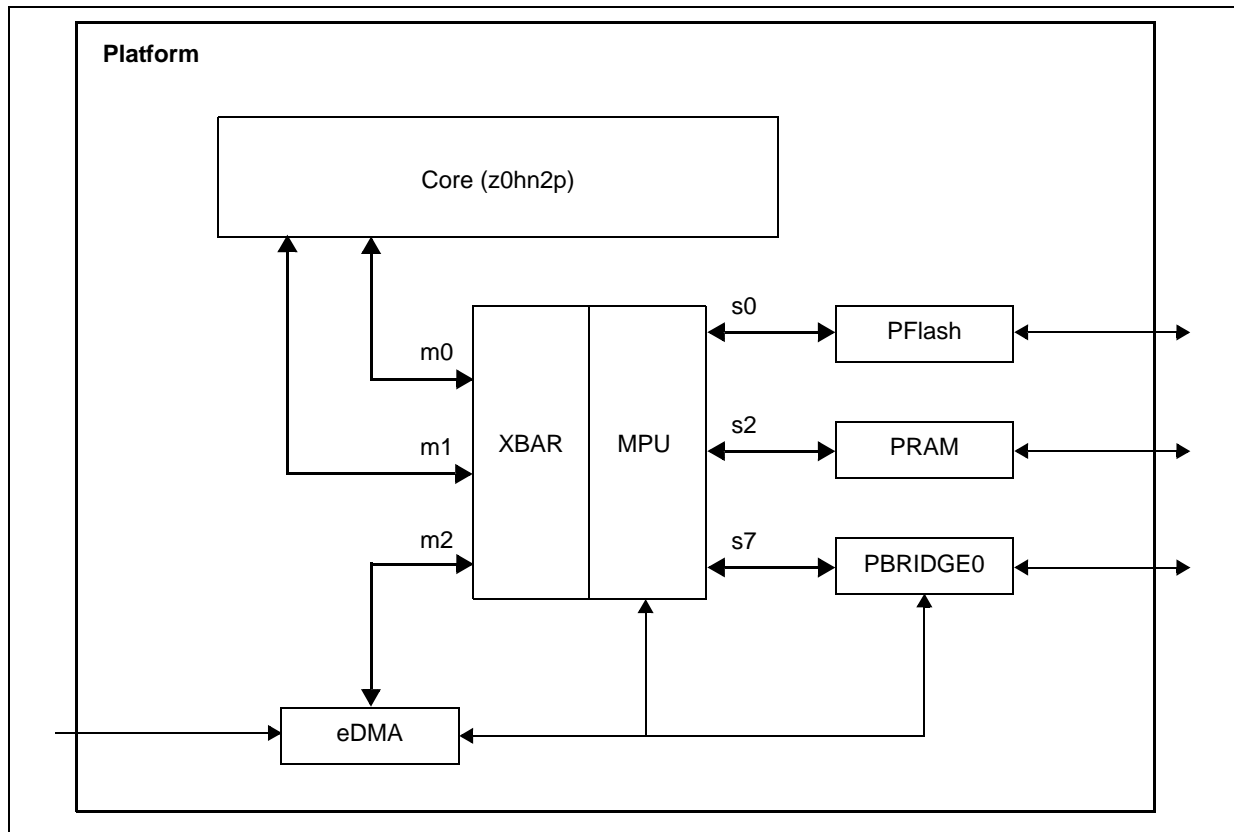


Figure 161. MPU block diagram

## 20.2 Features

The Memory Protection Unit implements a two-dimensional hardware array of memory region descriptors and the crossbar slave XBAR ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- Support for 8 memory region descriptors, each 128 bits in size
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
  - Access control definitions: 2 bus masters (processor cores) support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter only the access rights of a descriptor
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software. See

[Section 20.6.2: Putting it all together and AHB error terminations](#) for details and [Section 20.8: Application information](#) for an example.

- Support for 3 XBAR slave port connections: flash controller, system SRAM controller and peripherals bus
  - MPU hardware continuously monitors every XBAR slave port access using the preprogrammed memory region descriptors
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in all memory regions where it does hit. In the event of an access error, the XBAR reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device.
  - 64-bit error registers, one for each XBAR slave port, capture the last faulting address, attributes and “detail” information
- Global MPU enable/disable control bit provides a mechanism to easily load region descriptors during system startup or allow complete access rights during debug with the module disabled

## 20.3 Modes of operation

The MPU module does not support any special modes of operation. As a memory-mapped device located on the platform’s high-speed system bus, it responds based strictly on the memory addresses of the connected system buses. The peripheral bus is used to access the MPU’s programming model and the memory protection functions are evaluated on a reference-by-reference basis using the addresses from the XBAR system bus port(s).

Power dissipation is minimized when the MPU’s global enable/disable bit is cleared (MPU\_CESR[VLD] = 0).

## 20.4 External signal description

The MPU module does not include any external interface. The MPU’s internal interfaces include a peripheral bus connection for accessing the programming model and multiple connections to the address phase signals of the platform crossbar’s slave AHB ports. From a platform topology viewpoint, the MPU module appears to be directly connected “downstream” from the crossbar switch with interfaces to the XBAR slave ports.

## 20.5 Memory map and register description

The MPU module provides an IPS programming model mapped to an SPP-standard on-platform 16 KB space. The programming model is partitioned into three groups: control/status registers, the data structure containing the region descriptors and the alternate view of the region descriptor access control values.

The programming model can only be referenced using 32-bit (word) accesses. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an IPS error termination.

Finally, the programming model allocates space for an MPU definition with 8 region descriptors and up to 3 XBAR slave ports, like flash controller, system SRAM controller and peripheral bus.

## 20.5.1 Memory map

The MPU programming model map is shown in [Table 159](#).

**Table 159. MPU memory map**

Base address: 0xFFFF1_1000		
Address offset	Register	Location
0x000	MPU Control/Error Status Register (MPU_CESR)	<a href="#">on page 396</a>
0x004–0x00F	Reserved	
0x010	MPU Error Address Register, Slave Port 0 (MPU_EAR0)	<a href="#">on page 397</a>
0x014	MPU Error Detail Register, Slave Port 0 (MPU_EDR0)	<a href="#">on page 398</a>
0x018	MPU Error Address Register, Slave Port 1 (MPU_EAR1)	<a href="#">on page 397</a>
0x01C	MPU Error Detail Register, Slave Port 1 (MPU_EDR1)	<a href="#">on page 398</a>
0x020	MPU Error Address Register, Slave Port 2 (MPU_EAR2)	<a href="#">on page 397</a>
0x024	MPU Error Detail Register, Slave Port 2 (MPU_EDR2)	<a href="#">on page 398</a>
0x028–0x3FF	Reserved	
0x400	MPU Region Descriptor 0 (MPU_RGD0)	<a href="#">on page 399</a>
0x410	MPU Region Descriptor 1 (MPU_RGD1)	<a href="#">on page 399</a>
0x420	MPU Region Descriptor 2 (MPU_RGD2)	<a href="#">on page 399</a>
0x430	MPU Region Descriptor 3 (MPU_RGD3)	<a href="#">on page 399</a>
0x440	MPU Region Descriptor 4 (MPU_RGD4)	<a href="#">on page 399</a>
0x450	MPU Region Descriptor 5 (MPU_RGD5)	<a href="#">on page 399</a>
0x460	MPU Region Descriptor 6 (MPU_RGD6)	<a href="#">on page 399</a>
0x470	MPU Region Descriptor 7 (MPU_RGD7)	<a href="#">on page 399</a>
0x480–0x7FF	Reserved	
0x800	MPU RGD Alternate Access Control 0 (MPU_RGDAAC0)	<a href="#">on page 404</a>
0x804	MPU RGD Alternate Access Control 1 (MPU_RGDAAC1)	<a href="#">on page 404</a>
0x808	MPU RGD Alternate Access Control 2 (MPU_RGDAAC2)	<a href="#">on page 404</a>
0x80C	MPU RGD Alternate Access Control 3 (MPU_RGDAAC3)	<a href="#">on page 404</a>
0x810	MPU RGD Alternate Access Control 4 (MPU_RGDAAC4)	<a href="#">on page 404</a>
0x814	MPU RGD Alternate Access Control 5 (MPU_RGDAAC5)	<a href="#">on page 404</a>
0x818	MPU RGD Alternate Access Control 6 (MPU_RGDAAC6)	<a href="#">on page 404</a>
0x81C	MPU RGD Alternate Access Control 7 (MPU_RGDAAC7)	<a href="#">on page 404</a>

## 20.5.2 Register description

### 20.5.2.1 MPU Control/Error Status Register (MPU\_CESR)

The MPU\_CESR provides one byte of error status plus three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset: 0x000

Access: Read/Partial Write

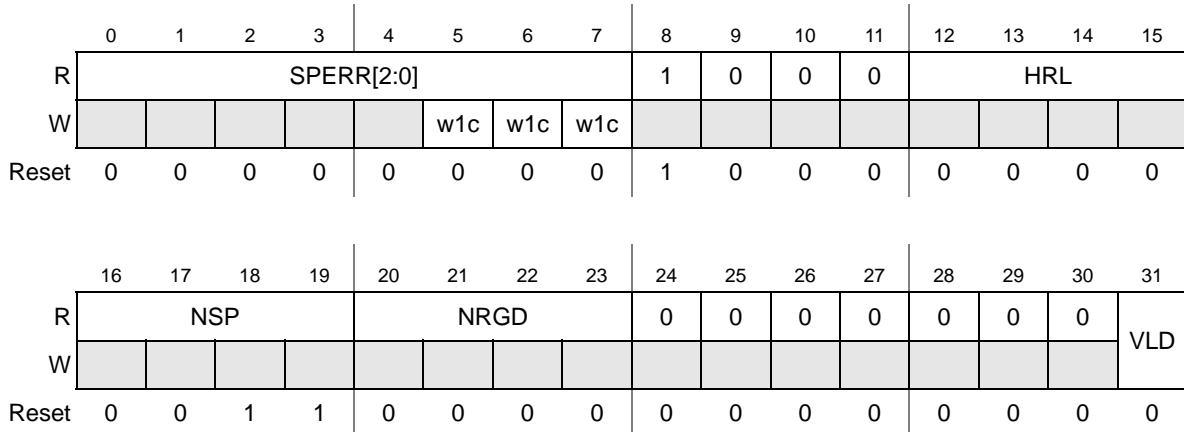


Figure 162. MPU Control/Error Status Register (MPU\_CESR)

Table 160. MPU\_CESR field descriptions

Field	Description
SPERRn	<p>Slave Port n Error, where the slave port number matches the bit number. See <a href="#">Table 161</a>.</p> <p>Each bit in this field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EARn and MPU_EDRn registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written as a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A “find first one” instruction (or equivalent) can be used to detect the presence of a captured error.</p> <p>0 The corresponding MPU_EARn/MPU_EDRn registers do not contain a captured error.                      1 The corresponding MPU_EARn/MPU_EDRn registers do contain a captured error.</p>
HRL	<p>Hardware Revision Level</p> <p>This field specifies the MPU's hardware and definition revision level. It can be read by software to determine the functional definition of the module.</p>
NSP	<p>Number of Slave Ports</p> <p>This field specifies the number of slave ports [1–8] connected to the MPU.</p>
NRGD	<p>Number of Region Descriptors</p> <p>This field specifies the number of region descriptors implemented in the MPU. The defined encodings include:</p> <p>0b0000 8 region descriptors                      0b0001 12 region descriptors                      0b0010 16 region descriptors</p>

**Table 160. MPU\_CESR field descriptions (continued)**

Field	Description
VLD	Valid This bit provides a global enable/disable for the MPU. 0 The MPU is disabled. 1 The MPU is enabled. While the MPU is disabled, all accesses from all bus masters are allowed.

**Table 161. SPERR implementation**

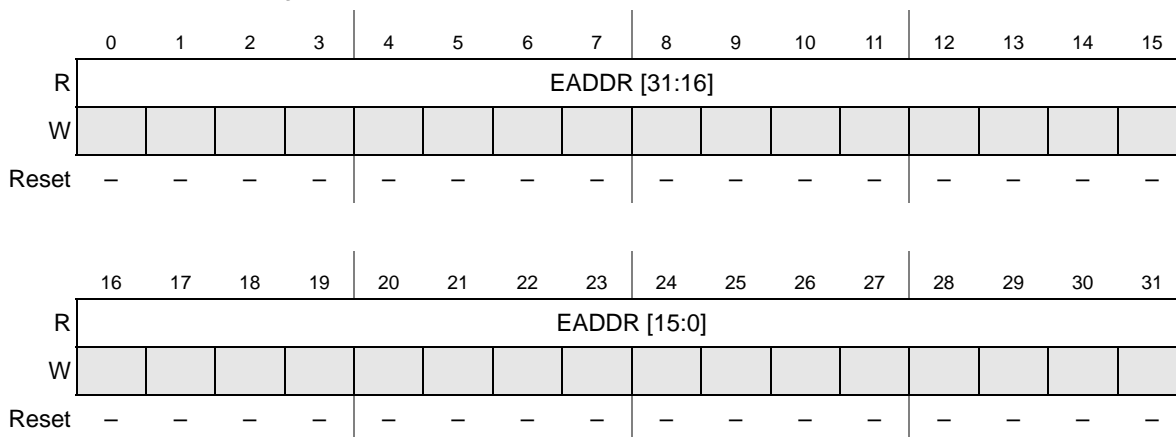
SPERR bit	Corresponding port
SPERR[0]	Flash memory controller slave port
SPERR[1]	System RAM controller slave port
SPERR[2]	IPS peripheral bus slave port

**20.5.2.2 MPU Error Address Register, Slave Port n (MPU\_EARn)**

When the MPU detects an access error on slave port n, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU\_CESR[SPERR] field set. Additional information about the faulting access is captured in the corresponding MPU\_EDRn register at the same time. Note this register and the corresponding MPU\_EDRn register contain the most recent access error; there are no hardware interlocks with the MPU\_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

Offsets: 0x010–0x020 (3 registers)

Access: Read



**Figure 163. MPU Error Address Register, Slave Port n (MPU\_EARn)**

**Table 162. MPU\_EARn field descriptions**

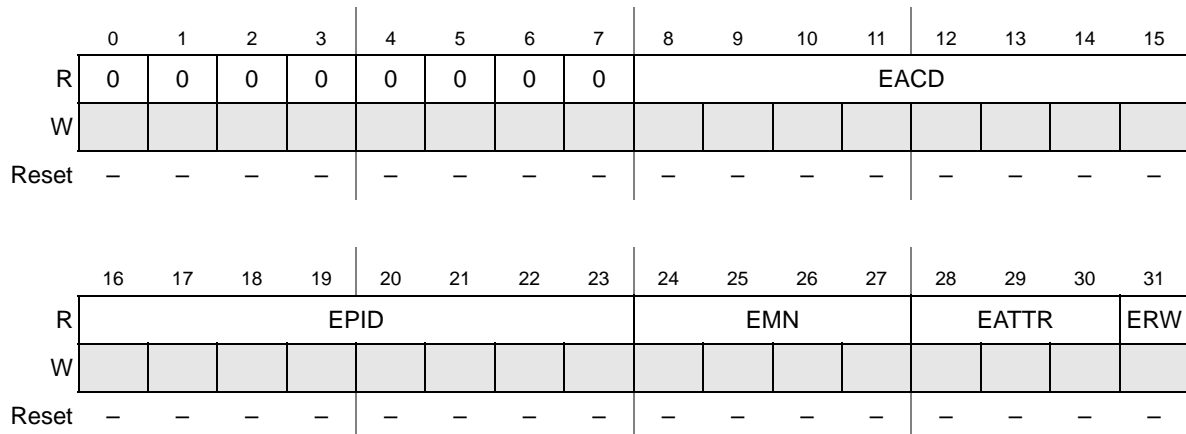
Field	Description
EADDR	Error Address This field is the reference address from slave port n that generated the access error.

**20.5.2.3 MPU Error Detail Register, Slave Port n (MPU\_EDRn)**

When the MPU detects an access error on slave port n, 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU\_CESR[SPERR] field set. Information on the faulting address is captured in the corresponding MPU\_EARn register at the same time. Note that this register and the corresponding MPU\_EARn register contain the most recent access error; there are no hardware interlocks with the MPU\_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

Offsets: 0x014–0x024 (3 registers)

Access: Read



**Figure 164. MPU Error Detail Register, Slave Port n (MPU\_EDRn)**

**Table 163. MPU\_EDRn field descriptions**

Field	Description
EACD	<p>Error Access Control Detail</p> <p>This field implements one bit per region descriptor and is an indication of the region descriptor hit logically ANDed with the access error indication. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field.</p> <p>If the MPU_EDRn register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits. If only a single EACD bit is set, then the protection error was caused by a single non-overlapping region descriptor. If two or more EACD bits are set, then the protection error was caused in an overlapping set of region descriptors.</p>
EPID	<p>Error Process Identification</p> <p>This field records the process identifier of the faulting reference. The process identifier is typically driven only by processor cores; for other bus masters, this field is cleared.</p>
EMN	<p>Error Master Number</p> <p>This field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.</p>

Table 163. MPU\_EDRn field descriptions (continued)

Field	Description
EATTR	<p>Error Attributes</p> <p>This field records attribute information about the faulting reference. The supported encodings are defined as:</p> <p>0b000 User mode, instruction access                      0b001 User mode, data access                      0b010 Supervisor mode, instruction access                      0b011 Supervisor mode, data access</p> <p>All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).</p>
ERW	<p>Error Read/Write</p> <p>This field signals the access type (read, write) of the faulting reference.</p> <p>0 Read                      1 Write</p>

20.5.2.4 MPU Region Descriptor n (MPU\_RGDn)

Each 128-bit (16 byte) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is the very essence of the operation of the Memory Protection Unit.

The region descriptors are organized sequentially in the MPU’s programming model and each of the four 32-bit words are detailed in the subsequent sections.

20.5.2.4.1 MPU Region Descriptor n, Word 0 (MPU\_RGDn.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor’s valid bit (see [Section 20.5.2.4.4: MPU Region Descriptor n, Word 3 \(MPU\\_RGDn.Word3\)](#) for more information).

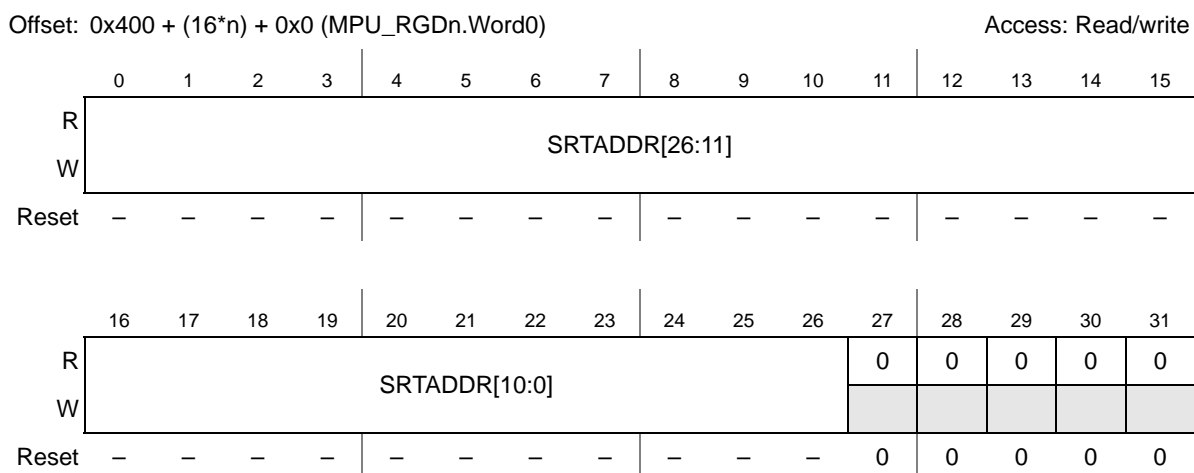


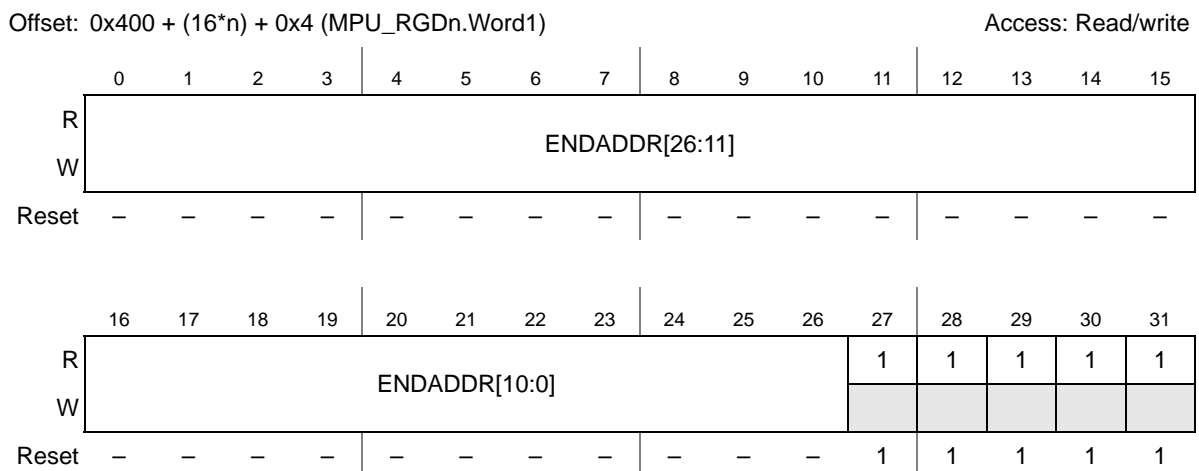
Figure 165. MPU Region Descriptor, Word 0 Register (MPU\_RGDn.Word0)

**Table 164. MPU\_RGDn.Word0 field descriptions**

Field	Description
SRTADDR	Start Address This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.

**20.5.2.4.2 MPU Region Descriptor n, Word 1 (MPU\_RGDn.Word1)**

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor’s valid bit (see [Section 20.5.2.4.4: MPU Region Descriptor n, Word 3 \(MPU\\_RGDn.Word3\)](#) for more information).



**Figure 166. MPU Region Descriptor, Word 1 Register (MPU\_RGDn.Word1)**

**Table 165. MPU\_RGDn.Word1 field descriptions**

Field	Description
ENDADDR	End Address This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR >= SRTADDR; it is software’s responsibility to properly load these region descriptor fields.

**20.5.2.4.3 MPU Region Descriptor n, Word 2 (MPU\_RGDn.Word2)**

The third word of the MPU region descriptor defines the access control rights of the memory region. The access control privileges are dependent on two broad classifications of bus masters. Bus masters 0–3 are typically reserved for processor cores and the corresponding access control is a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. Bus masters 4–7 are typically reserved for data movement engines and their capabilities are limited to separate read and write permissions. For these fields, the bus master number refers to the logical master number defined as the XBAR hmaster[3:0] signal.



For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (*r*) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (*w*) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (*x*) permission refers to the ability to read the referenced memory address using an instruction fetch.

The evaluation logic defines the processor access type based on multiple AHB signals, as *hwrite* and *hprot[1:0]*.

For non-processor data movement engines (bus masters 4–7), the evaluation logic simply uses *hwrite* to determine if the access is a read or write.

Writes to this word clear the region descriptor’s valid bit (see [Section 20.5.2.4.4: MPU Region Descriptor n, Word 3 \(MPU\\_RGDn.Word3\)](#) for more information). Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU\_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU\_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor’s valid bit.

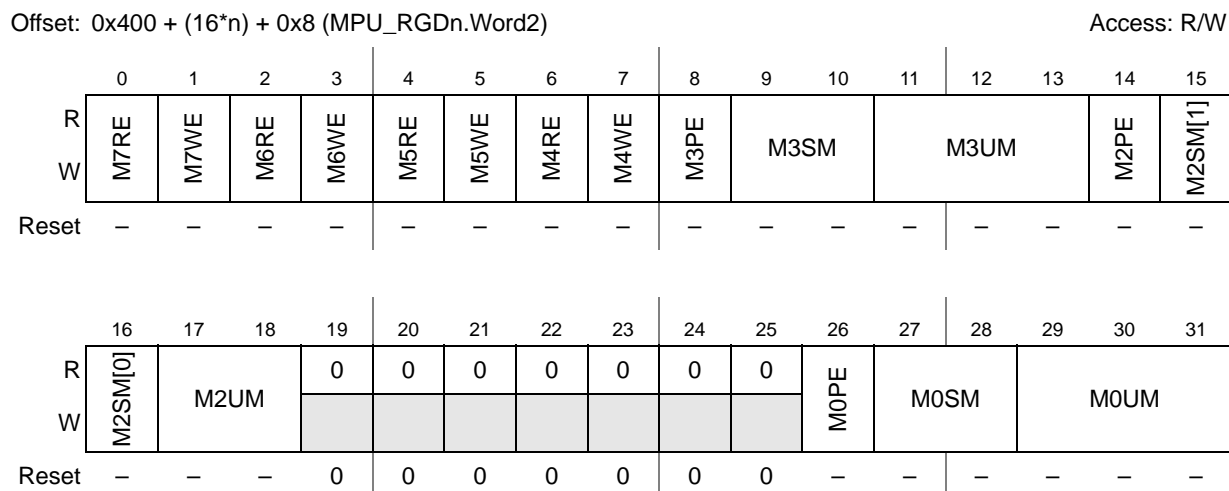


Figure 167. MPU Region Descriptor, Word 2 Register (MPU\_RGDn.Word2)

Table 166. MPU\_RGDn.Word2 field descriptions

Field	Description
M7RE	Bus master 7 read enable If set, this flag allows bus master 7 to perform read operations. If cleared, any attempted read by bus master 7 terminates with an access error and the read is not performed.
M7WE	Bus master 7 write enable If set, this flag allows bus master 7 to perform write operations. If cleared, any attempted write by bus master 7 terminates with an access error and the write is not performed.

**Table 166. MPU\_RGDn.Word2 field descriptions (continued)**

Field	Description
M6RE	<p>Bus master 6 read enable</p> <p>If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.</p>
M6WE	<p>Bus master 6 write enable</p> <p>If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed.</p>
M5RE	<p>Bus master 5 read enable</p> <p>If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.</p>
M5WE	<p>Bus master 5 write enable</p> <p>If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed.</p>
M4RE	<p>Bus master 4 read enable</p> <p>If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.</p>
M4WE	<p>Bus master 4 write enable</p> <p>If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.</p>
M3PE	<p>Bus master 3 process identifier enable</p> <p>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.</p>
M3SM	<p>Bus master 3 supervisor mode access control</p> <p>This field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as:</p> <p>0b00 r, w, x = read, write and execute allowed                      0b01 r, -, x = read and execute allowed, but no write                      0b10 r, w, - = read and write allowed, but no execute                      0b11 Same access controls as that defined by M3UM for user mode</p>
M3UM	<p>Bus master 3 user mode access control</p> <p>This field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.</p>
M2PE	<p>Bus master 2 process identifier enable</p> <p>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.</p>
M2SM	<p>Bus master 2 supervisor mode access control</p> <p>This field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as:</p> <p>0b00 r, w, = read and write allowed                      0b01 r = read allowed, but no write                      0b10 r, w = read and write allowed                      0b11 Same access controls as that defined by M2UM for user mode</p>

**Table 166. MPU\_RGDn.Word2 field descriptions (continued)**

Field	Description
M2UM	Bus master 2 user mode access control This field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of two independent bits, enabling read and write permissions: {r,w}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus master 0 process identifier enable If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M0SM	Bus master 0 supervisor mode access control This field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 0b00 r, w, x = read, write and execute allowed 0b01 r, -, x = read and execute allowed, but no write 0b10 r, w, - = read and write allowed, but no execute 0b11 Same access controls as that defined by M0UM for user mode
M0UM	Bus master 0 user mode access control This field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

**20.5.2.4.4 MPU Region Descriptor n, Word 3 (MPU\_RGDn.Word3)**

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Since the region descriptor is a 128-bit entity, there are potential coherency issues as this structure is being updated since multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU\_RGDn.Word0, then MPU\_RGDn.Word1,... and finally MPU\_RGDn.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU\_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU\_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor's valid bit.

Offset: 0x400 + (16\*n) + 0xC (MPU\_RGDn.Word3)

Access: Read/write

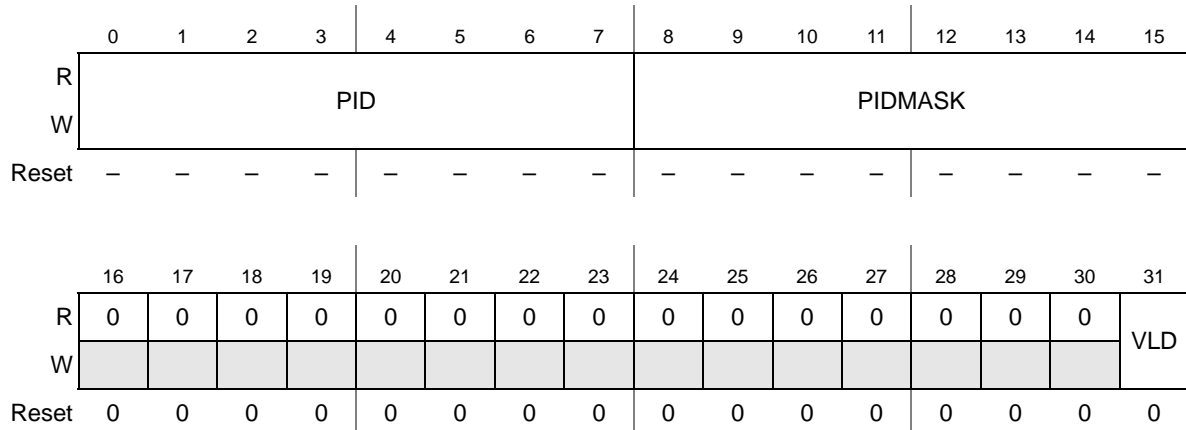


Figure 168. MPU Region Descriptor, Word 3 Register (MPU\_RGDn.Word3)

Table 167. MPU\_RGDn.Word3 field descriptions

Field	Description
PID	Process Identifier This field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set.
PIDMASK	Process Identifier Mask This field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, then the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see <a href="#">Section 20.6.1.1: Access evaluation – Hit determination</a> .
VLD	Valid This bit signals the region descriptor is valid. Any write to MPU_RGDn.Word{0,1,2} clears this bit, while a write to MPU_RGDn.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid 1 Region descriptor is valid

### 20.5.2.5 MPU Region Descriptor Alternate Access Control n (MPU\_RGDAACn)

As noted in [Section 20.5.2.4.3: MPU Region Descriptor n, Word 2 \(MPU\\_RGDn.Word2\)](#), it is expected that since system software may adjust only the access controls within a region descriptor (MPU\_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is desired. If only the access controls are being updated, this operation should be performed by writing to MPU\_RGDAACn (Alternate Access Control n) as stores to these locations do not affect the descriptor’s valid bit.

The memory address therefore provides an alternate location for updating MPU\_RGDn.Word2.

Offset: 0x800 + (4\*n) (MPU\_RGDAACn)

Access: Read/write

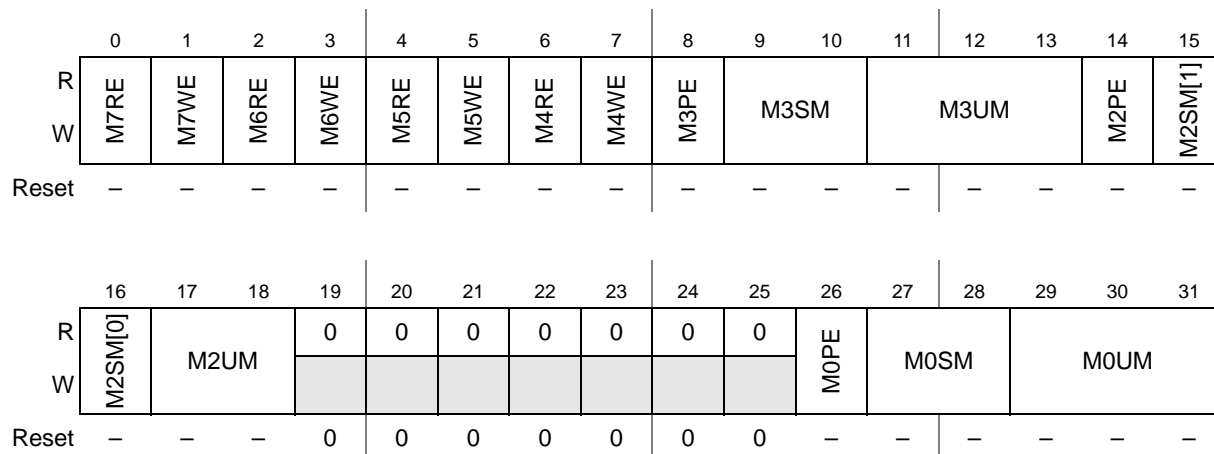


Figure 169. MPU RGD Alternate Access Control n (MPU\_RGDAACn)

Since the MPU\_RGDAACn register is simply another memory mapping for MPU\_RGDn.Word2, the field definitions shown in Table 168 are identical to those presented in Table 166.

Table 168. MPU\_RGDAACn field descriptions

Field	Description
M7RE	Bus master 7 read enable. If set, this flag allows bus master 7 to perform read operations. If cleared, any attempted read by bus master 7 terminates with an access error and the read is not performed.
M7WE	Bus master 7 write enable If set, this flag allows bus master 7 to perform write operations. If cleared, any attempted write by bus master 7 terminates with an access error and the write is not performed.
M6RE	Bus master 6 read enable If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.
M6WE	Bus master 6 write enable If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed.
M5RE	Bus master 5 read enable If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.
M5WE	Bus master 5 write enable If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed.
M4RE	Bus master 4 read enable If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.
M4WE	Bus master 4 write enable If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.



**Table 168. MPU\_RGDAACn field descriptions (continued)**

Field	Description
M3PE	<p>Bus master 3 process identifier enable</p> <p>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDN.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.</p>
M3SM	<p>Bus master 3 supervisor mode access control</p> <p>This field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as:</p> <p>0b00 r, w, x = read, write and execute allowed                      0b01 r, -, x = read and execute allowed, but no write                      0b10 r, w, - = read and write allowed, but no execute                      0b11 Same access controls as that defined by M3UM for user mode</p>
M3UM	<p>Bus master 3 user mode access control</p> <p>This field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.</p>
M2PE	<p>Bus master 2 process identifier enable</p> <p>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDN.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.</p>
M2SM	<p>Bus master 2 supervisor mode access control</p> <p>This field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as:</p> <p>0b00 r, w, = read and write allowed                      0b01 r = read allowed, but no write                      0b10 r, w = read and write allowed                      0b11 Same access controls as that defined by M2UM for user mode</p>
M2UM	<p>Bus master 2 user mode access control</p> <p>This field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.</p>
M0PE	<p>Bus master 0 process identifier enable</p> <p>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDN.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.</p>
M0SM	<p>Bus master 0 supervisor mode access control</p> <p>This field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as:</p> <p>0b00 r, w, x = read, write and execute allowed                      0b01 r, -, x = read and execute allowed, but no write                      0b10 r, w, - = read and write allowed, but no execute                      0b11 Same access controls as that defined by M0UM for user mode</p>

Table 168. MPU\_RGDAACn field descriptions (continued)

Field	Description
MOUM	Bus master 0 user mode access control This field defines the access controls for bus master 0 when operating in user mode. The MOUM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

## 20.6 Functional description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated bus cycles.

### 20.6.1 Access evaluation macro

As previously discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in [Figure 170](#), the access evaluation macro inputs the system bus address phase signals and the contents of a region descriptor (RGDn) and performs two major functions: region hit determination (hit\_b) and detection of an access protection violation (error).

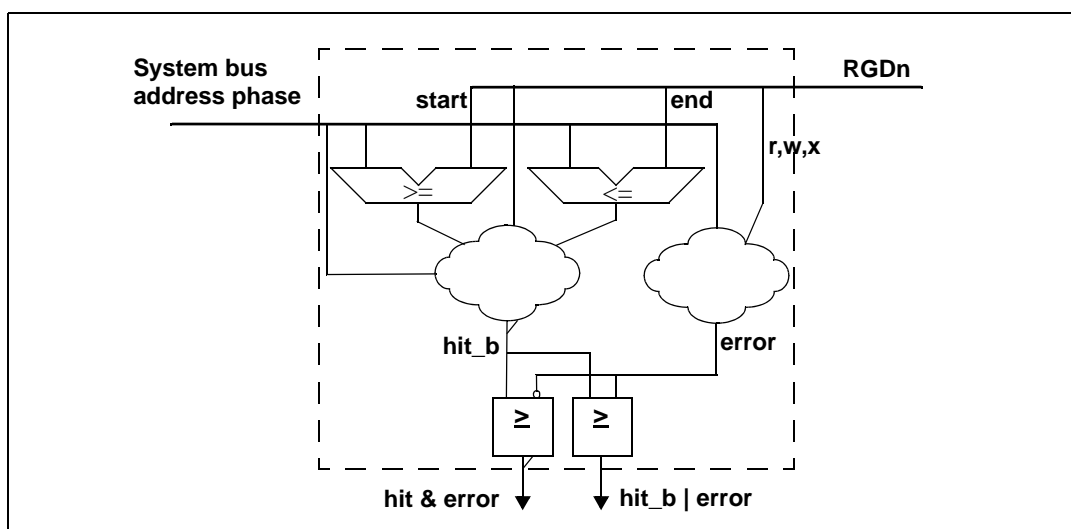


Figure 170. MPU access evaluation macro

[Figure 170](#) is not intended to be a schematic of the actual access evaluation macro, but rather a generalized block diagram showing the major functions included in this logic block.

#### 20.6.1.1 Access evaluation – Hit determination

To evaluate the region hit determination, the MPU uses two magnitude comparators in conjunction with the contents of a region descriptor: the current access must be included between the region's "start" and "end" addresses and simultaneously the region's valid bit must be active.

Recall there are no hardware checks to verify that region's "end" address is greater than region's "start" address, and it is software's responsibility to properly load appropriate values into these fields of the region descriptor.

In addition to this, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. In order to generate the pid\_hit indication: the current PID with its PIDMASK must be equal to the region's PID with its PIDMASK. Also the process identifier enable is taken into account in this comparison so that the MPU forces the pid\_hit term to be asserted in the case of AHB bus master doesn't provide its process identifier.

### 20.6.1.2 Access evaluation – Privilege violation determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown in [Table 169](#).

**Table 169. Protection violation definition**

Description	Inputs			Output
	eff_rgd[r]	eff_rgd[w]	eff_rgd[x]	Protection violation?
inst fetch read	—	—	0	yes, no x permission
inst fetch read	—	—	1	no, access is allowed
data read	0	—	—	yes, no r permission
data read	1	—	—	no, access is allowed
data write	—	0	—	yes, no w permission
data write	—	1	—	no, access is allowed

As shown in [Figure 170](#), the output of the protection violation logic is the error signal.

The access evaluation macro then uses the hit\_b and error signals to form two outputs. The combined (hit\_b | error) signal is used to signal the current access is not allowed and (~hit\_b & error) is used as the input to MPU\_EDRn (error detail register) in the event of an error.

### 20.6.2 Putting it all together and AHB error terminations

For each XBAR slave port being monitored, the MPU performs a reduction-AND of all the individual (hit\_b | error) terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in any region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, then a protection error is reported.
3. If the access hits in multiple (overlapping) regions and all regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to permission granting over access denying for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see [Section 20.8: Application information](#).



In event of a protection error, the MPU requires two distinct actions:

1. Intercepting the error during the address phase (first cycle out of two) and cancelling the transaction before it is seen by the slave device
2. Performing the required logic functions to force the standard 2-cycle AHB error response to properly terminate the bus transaction and then providing the right values to the crossbar switch to commit the transaction to other portions of the platform.

If, instead, the access is allowed, then the MPU simply passes all “original” signals to the slave device. In this case, from a functionality point of view, the MPU is fully transparent.

## 20.7 Initialization information

The reset state of MPU\_CESR[VLD] disables the entire module. Recall that, while the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU\_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU\_RGDn) is loaded at system startup, including the setting of the MPU\_RGDn.Word3[VLD] bits, before MPU\_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. Recall if a memory reference does not hit in any region descriptor, the attempted access is terminated with an error.

## 20.8 Application information

In an operational system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGDn, it would typically be performed using four 32-bit word writes. As discussed in [Section 20.5.2.4.4: MPU Region Descriptor n, Word 3 \(MPU\\_RGDn.Word3\)](#), the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed simply by clearing MPU\_RGDn.Word3[VLD].
2. If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU\_RGDAACn) would typically be performed. Recall writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the IPS write completes.
3. If the region's start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU\_RGDn.Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses respectively and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
4. Typically, references to the MPU's programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would

be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.

When the MPU detects an access error, the current bus cycle is terminated with an error response and information on the faulting reference captured in the MPU\_EARn and MPU\_EDRn registers. The error-terminated bus cycle typically initiates some type of error response in the originating bus master. For example, the CPU errors will generate a core exception, whereas the DMA errors will generate a MPU (external) interrupt. It is important to highlight that in case of DMA access violations the core will continue to run, but if a core violation occurs the system will stop. In any event, the processor can retrieve the captured error address and detail information simply by reading the MPU\_E{A,D}Rn registers. Information on which error registers contain captured fault data is signaled by MPU\_CESR[SPERR].

## 21 System Integration Unit Lite (SIUL)

### 21.1 Introduction

This chapter describes the System Integration Unit Lite (SIUL), which is used for the management of the pads and their configuration. It controls the multiplexing of the alternate functions used on all pads as well as being responsible for the management of the external interrupts to the device.

### 21.2 Overview

The System Integration Unit Lite (SIUL) controls the MCU pad configuration, ports, general-purpose input and output (GPIO) signals and external interrupts with trigger event configuration. [Figure 171](#) provides a block diagram of the SIUL and its interfaces to other system components.

The module provides the capability to configure, read, and write to the device's general-purpose I/O pads that can be configured as either inputs or outputs.

- When a pad is configured as an input, the state of the pad (logic high or low) is obtained by reading an associated data input register.
- When a pad is configured as an output, the value driven onto the pad is determined by writing to an associated data output register. Enabling the input buffers when a pad is configured as an output allows the actual state of the pad to be read.
- To enable monitoring of an output pad value, the pad can be configured as both output and input so the actual pad value can be read back and compared with the expected value.

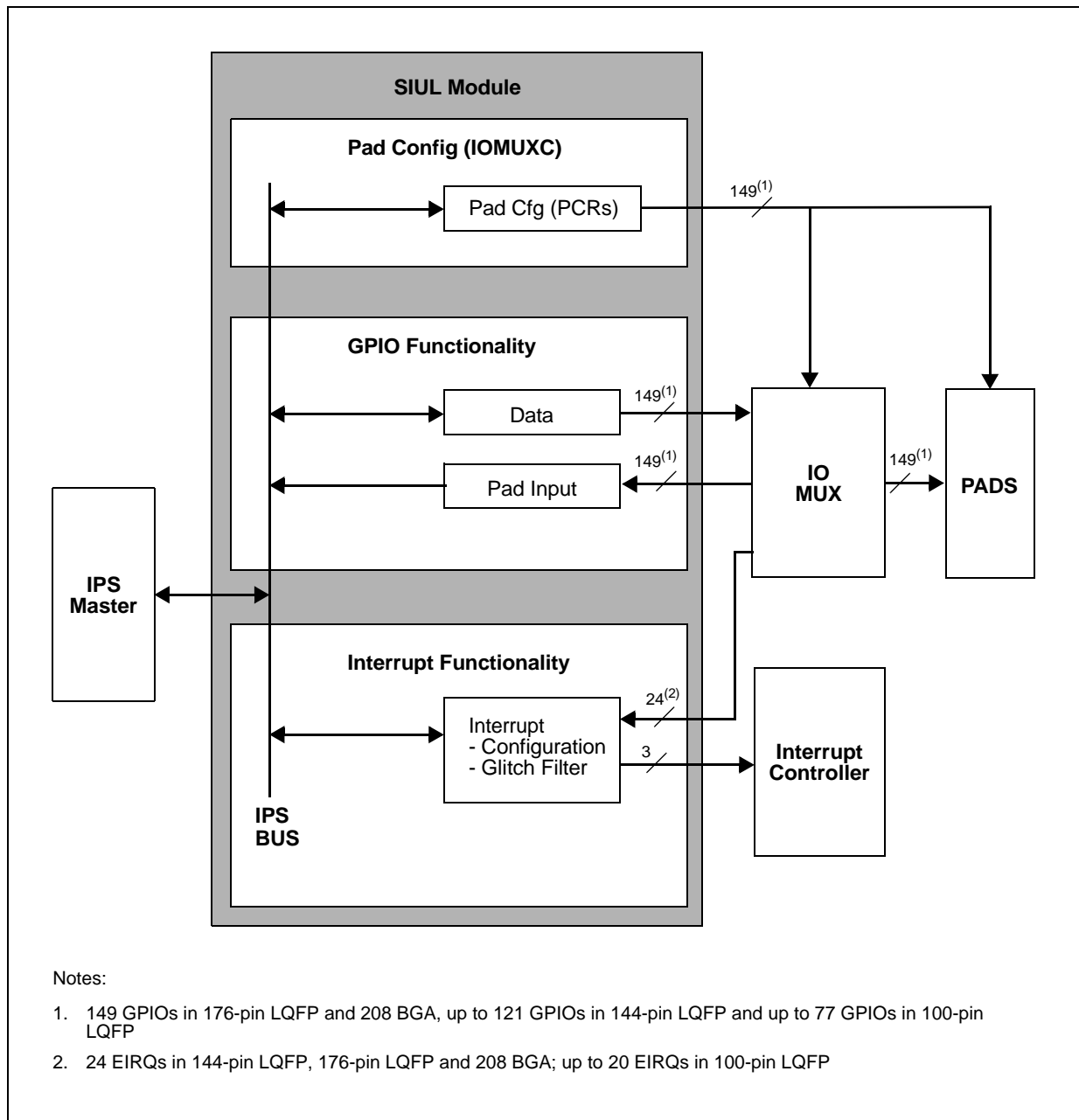


Figure 171. System Integration Unit Lite block diagram

## 21.3 Features

The System Integration Unit Lite supports these distinctive features:

- GPIO
  - GPIO function on up to 149 I/O pins
  - Dedicated input and output registers for most GPIO pins<sup>(o)</sup>
- External interrupts
  - 3 interrupt vectors dedicated to 24 external interrupts
  - 24 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

## 21.4 External signal description

Most device pads support multiple device functions. Pad configuration registers are provided to enable selection between GPIO and other signals. These other signals, also referred to as alternate functions, are typically peripheral functions.

GPIO pads are grouped in “ports”, with each port containing up to 16 pads. With appropriate configuration, all pins in a port can be read or written to in parallel with a single R/W access.

*Note:* In order to use GPIO port functionality, all pads in the port must be configured as GPIO rather than as alternate functions.

Table 170 lists the external pins configurable via the SIUL.

**Table 170. SIUL signal properties**

GPIO[0:148] category	Name	I/O direction	Function
System configuration	GPIO[0:148] <sup>(1)</sup>	I/O <sup>(2)</sup>	General-purpose input/output
External interrupt	PA[3], PA[6:8], PA[11:12], PA[14], PC[2:5], PC[12], PC[14:15], PE[2], PE[4], PE[6:7], PE[10], PE[12], PE[14], PF[15], PG[1], PG[8] <sup>(3)</sup>	Input	Pins with External Interrupt Request functionality. Please see the signal description chapter of this reference manual for details.

1. GPIO[0–26], GPIO[28–59] and GPIO[61–122] in 144-pin LQFP; GPIO[0–26], GPIO[28–59], GPIO[61–76] and GPIO[121–122] in 100-pin LQFP
2. Most, but not all GPIO pads can be configured as inputs or outputs but some, e.g., analog pins with GPIO function, are only configurable as inputs.
3. PE[14], PF[15], PG[1] and PG[8] not available in 100-pin LQFP

<sup>o</sup> Some device pins, e.g., analog pins, do not have both input and output functionality.

## 21.4.1 Detailed signal descriptions

### 21.4.1.1 General-purpose I/O pins (GPIO[0:148])<sup>(p)</sup>

The GPIO pins provide general-purpose input and output function. The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an input (GPDIn<sub>n</sub>) or output (GPDO<sub>n</sub>) register.

### 21.4.1.2 External interrupt request input pins (EIRQ[0:23])<sup>(q)</sup>

The EIRQ[0:23] pins are connected to the SIUL inputs. Rising- or falling-edge events are enabled by setting the corresponding bits in the SIUL\_IREER or the SIUL\_IFEER register.

---

p. GPIO[0–26], GPIO[28–59] and GPIO[61–122] in 144-pin LQFP; GPIO[0–26], GPIO[28–59], GPIO[61–76] and GPIO[121–122] in 100-pin LQFP

q. EIRQ[0:11] plus EIRQ[16:23] in 100-pin LQFP

## 21.5 Memory map and register description

This section provides a detailed description of all registers accessible in the SIUL module.

### 21.5.1 SIUL memory map

[Table 171](#) gives an overview of the SIUL registers implemented.

**Table 171. SIUL memory map**

Base address: 0xC3F9_0000		
Address offset	Register	Location
0x0000	Reserved	
0x0004	MCU ID Register #1 (MIDR1)	<a href="#">on page 417</a>
0x0008		<a href="#">on page 418</a>
0x000C–0x0013	Reserved	
0x0014	Interrupt Status Flag Register (ISR)	<a href="#">on page 419</a>
0x0018	Interrupt Request Enable Register (IRER)	<a href="#">on page 420</a>
0x001C–0x0027	Reserved	
0x0028	Interrupt Rising-Edge Event Enable Register (IREER)	<a href="#">on page 420</a>
0x002C	Interrupt Falling-Edge Event Enable Register (IFEER)	<a href="#">on page 421</a>
0x0030	Interrupt Filter Enable Register (IFER)	<a href="#">on page 422</a>
0x0034–0x003F	Reserved	
0x0040–0x0168	Pad Configuration Registers (PCR0–PCR148) <sup>(1)</sup>	<a href="#">on page 423</a>
0x016A–0x04FF	Reserved	
0x0500–0x053C	Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI60_63)	<a href="#">on page 425</a>
0x0540–0x05FF	Reserved	
0x0600–0x06A0	GPIO Pad Data Output Registers (GPDO0_3–GPDO148_151) <sup>(2)</sup>	<a href="#">on page 430</a>
0x06A4–0x07FF	Reserved	
0x0800–0x08A0	GPIO Pad Data Input Registers (GPDIO_3–GPDIO148_151) <sup>(3)</sup>	<a href="#">on page 431</a>
0x08A4–0x0BFF	Reserved	
0x0C00–0x0C10	Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO4)	<a href="#">on page 431</a>
0x0C14–0x0C3F	Reserved	
0x0C40–0x0C50	Parallel GPIO Pad Data In Registers (PGPDI0 – PGPDI4)	<a href="#">on page 432</a>
0x0C54–0x0C7F	Reserved	
0x0C80–0x0CA4	Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO9)	<a href="#">on page 433</a>
0x0CA8–0x0FFF	Reserved	

Table 171. SIUL memory map (continued)

Base address: 0xC3F9_0000		
Address offset	Register	Location
0x1000–0x105C	Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23) <sup>(4)</sup>	<i>on page 434</i>
0x1060–0x107C	Reserved	
0x1080	Interrupt Filter Clock Prescaler Register (IFCPR)	<i>on page 435</i>
0x1084–0x3FFF	Reserved	

1. PCR0–26, PCR28–59 and PCR61–122 in 144-pin LQFP; PCR0–26, PCR28–59, PCR61–76 and PCR121–122 in 100-pin LQFP—all remaining registers are reserved.
2. GPDO0–26, GPDO28–59 and GPDO61–122 in 144-pin LQFP; GPDO0–26, GPDO28–59, GPDO61–76 and GPDO121–122 in 100-pin LQFP—all remaining registers are reserved.
3. GPDIO0–26, GPDIO28–59 and GPDIO61–122 in 144-pin LQFP; GPDIO0–26, GPDIO28–59, GPDIO61–76 and GPDIO121–122 in 100-pin LQFP—all remaining registers are reserved.
4. IFMC0–11 plus IFMC16–23 in 100-pin LQFP—all remaining registers are reserved.

*Note:* A transfer error will be issued when trying to access completely reserved register space.

## 21.5.2 Register protection

Individual registers in System Integration Unit Lite can be protected from accidental writes using the Register Protection module. The following registers can be protected:

- Interrupt Request Enable Register (IRER)
- Interrupt Rising-Edge Event Enable Register (IREER)
- Interrupt Falling-Edge Event Enable Register (IFEER)
- Interrupt Filter Enable Register (IFER),
- Pad Configuration Registers (PCR0–PCR148). Note that only the following registers can be protected:
  - PCR[0:15] (Port A)
  - PCR[16:19] (Port B[0:3])
  - PCR[34:47] (Port C[2:15])
- Pad Selection for Multiplexed Inputs Registers (PSMI0\_3–PSMI60\_63)
- Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23). Note that only IFMC[0:15] can be protected.
- Interrupt Filter Clock Prescaler Register (IFCPR)

See the “Register Under Protection” appendix for more details.

## 21.5.3 Register descriptions

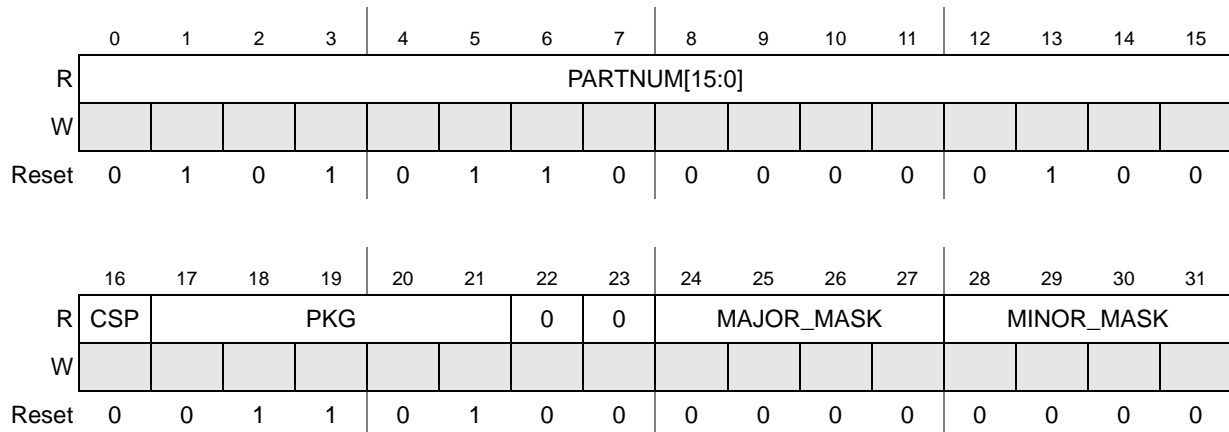
### 21.5.3.1 MCU ID Register #1 (MIDR1)

This register holds identification information about the device.



Offset: 0x0004

Access: Read



**Figure 172. MCU ID Register #1 (MIDR1)**

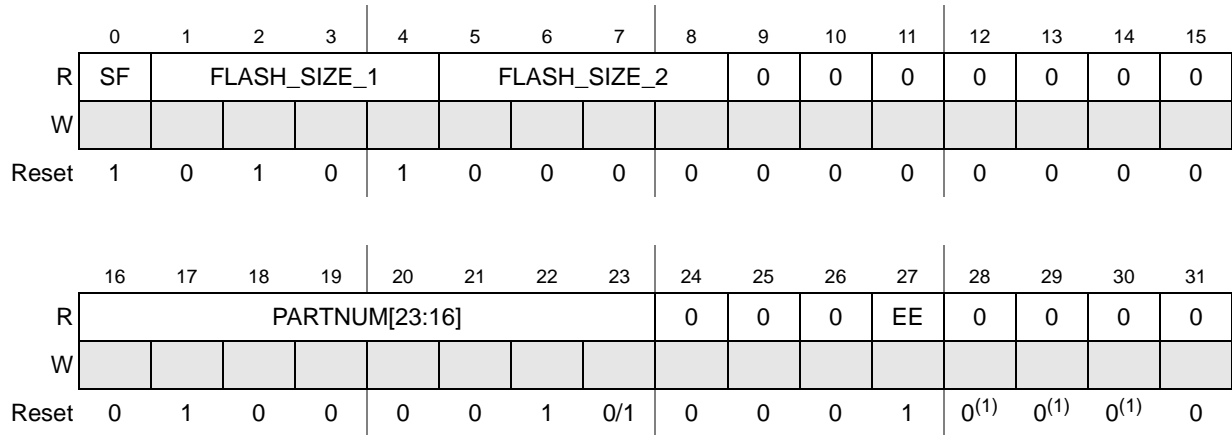
**Table 172. MIDR1 field descriptions**

Field	Description
PARTNUM[15:0]	MCU Part Number, lower 16 bits Device part number of the MCU. 0101_0110_0000_0001: 128 KB 0101_0110_0000_0010: 256 KB 0101_0110_0000_0011: 320/384 KB 0101_0110_0000_0100: 512 KB 0101_0110_0000_0101: 768 KB 0101_0110_0000_0110: 1 MB 0101_0110_0000_0111: 1.5 MB For the full part number this field needs to be combined with MIDR2[PARTNUM[23:16]].
CSP	Always reads back 0
PKG	Package Settings Can be read by software to determine the package type that is used for the particular device as described below. Any values not explicitly specified are reserved. 0b01001: 100-pin LQFP 0b01101: 144-pin LQFP 0b10000: 208 BGA 0b10001: 176-pin LQFP
MAJOR_MASK	Major Mask Revision Counter starting at 0x0. Incremented each time there is a resynthesis.
MINOR_MASK	Minor Mask Revision Counter starting at 0x0. Incremented each time a mask change is done.

**21.5.3.2 MCU ID Register #2 (MIDR2)**

Offset: 0x0008

Access: Read



**Figure 173. MCU ID Register #2 (MIDR2)**

- 1. Static bit fixed in hardware

**Table 173. MIDR2 field descriptions**

Field	Description
SF	Manufacturer 0 Reserved 1 ST
FLASH_SIZE_1	Coarse granularity for Flash memory size Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2 0011 128 KB 0100 256 KB 0101 512 KB 0110 1 MB
FLASH_SIZE_2	Fine granularity for Flash memory size Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2 0000 0 x (FLASH_SIZE_1 / 8) 0010 2 x (FLASH_SIZE_1 / 8) 0100 4 x (FLASH_SIZE_1 / 8)
PARTNUM [23:16]	MCU Part Number, upper 8 bits containing the ASCII character within the MCU part number 0x42h: Character 'B' (Body controller) 0x43h: Character 'C' (Gateway)  For the full part number this field needs to be combined with MIDR1[PARTNUM[15:0]].
EE	Data Flash present 0 No Data Flash is present 1 Data Flash is present

**21.5.3.3 Interrupt Status Flag Register (ISR)**

This register holds the interrupt flags.



Offset: 0x0014

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	EIF[23:16] <sup>(1)</sup>							
W									w1c							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF[15:0] <sup>(1)</sup>															
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 174. Interrupt Status Flag Register (ISR)**

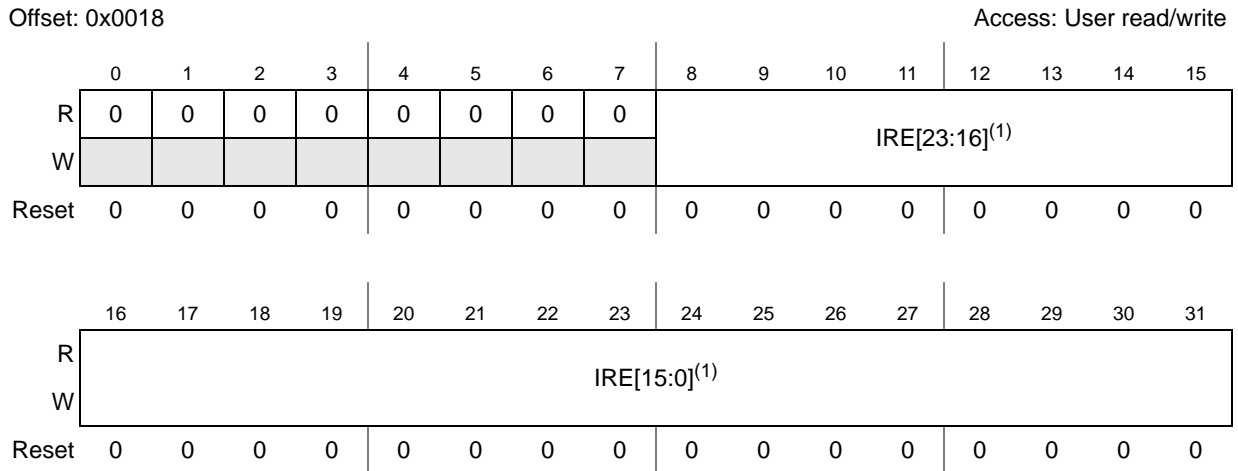
1. 20 flags in 100-pin LQFP: EIF[23:16] plus EIF[11:0] (register bits 16–19 reserved).

**Table 174. ISR field descriptions**

Field	Description
EIF[x]	External Interrupt Status Flag x This flag can be cleared only by writing a '1'. Writing a '0' has no effect. If enabled (IRER[x]), EIF[x] causes an interrupt request. 0 No interrupt event has occurred on the pad 1 An interrupt event as defined by IREER[x] and IFEER[x] has occurred

### 21.5.3.4 Interrupt Request Enable Register (IRER)

This register is used to enable the interrupt messaging to the interrupt controller.



**Figure 175. Interrupt Request Enable Register (IRER)**

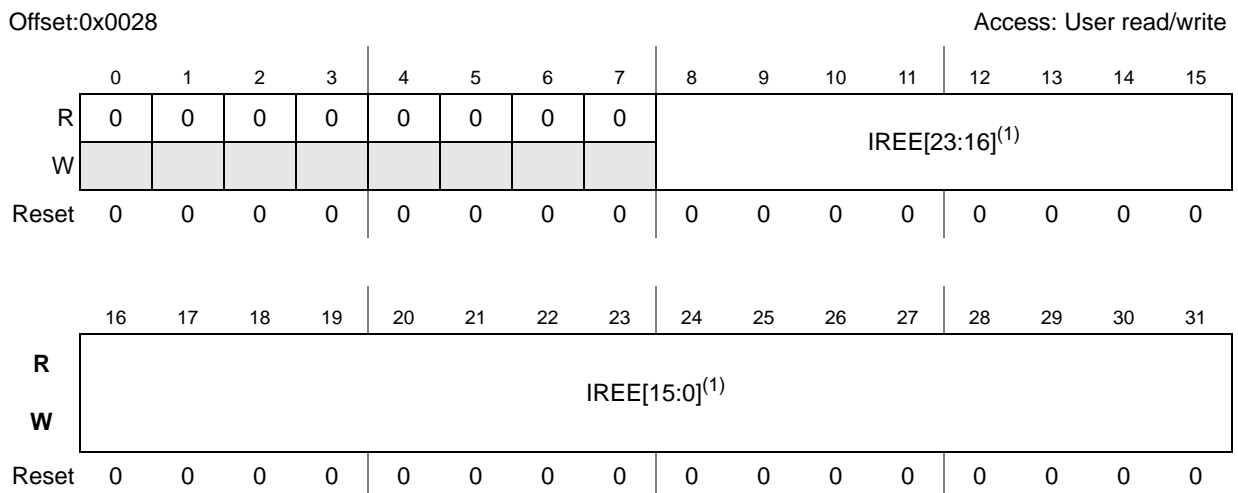
1. 20 enable requests in 100-pin LQFP: IRE[23:16] plus IRE[11:0] (register bits 16–19 reserved).

**Table 175. IRER field descriptions**

Field	Description
IRE[x]	External Interrupt Request Enable x 0 Interrupt requests from the corresponding ISR[EIF[x]] bit are disabled. 1 Interrupt requests from the corresponding ISR[EIF[x]] bit are enabled.

### 21.5.3.5 Interrupt Rising-Edge Event Enable Register (IREER)

This register is used to enable rising-edge triggered events on the corresponding interrupt pads.



**Figure 176. Interrupt Rising-Edge Event Enable Register (IREER)**

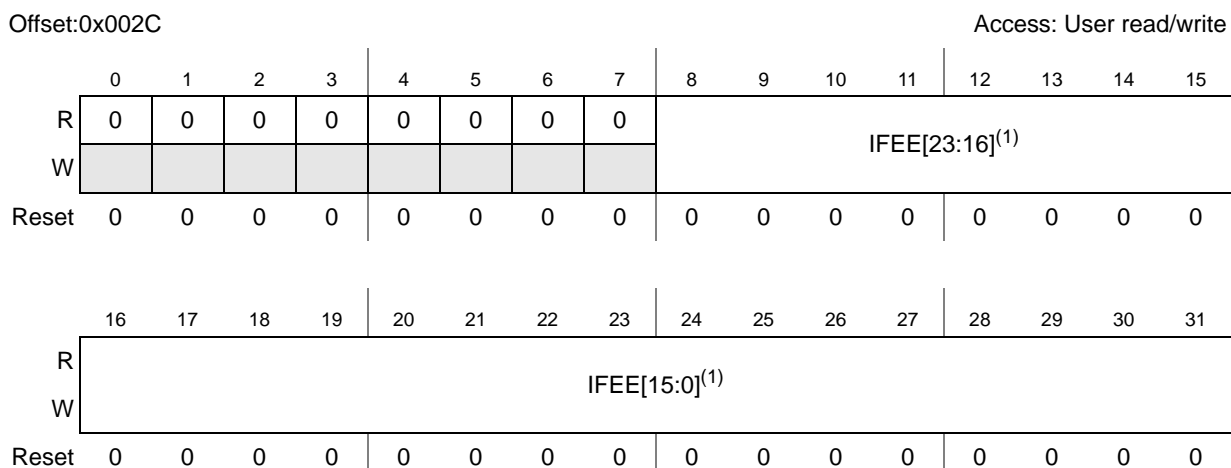
1. 20 enable events in 100-pin LQFP: IREE[23:16] plus IREE[11:0] (register bits 16–19 reserved).

**Table 176. IREER field descriptions**

Field	Description
IREE[x]	Enable rising-edge events to cause the ISR(EIF[x]) bit to be set. 0 Rising-edge event is disabled 1 Rising-edge event is enabled

**21.5.3.6 Interrupt Falling-Edge Event Enable Register (IFEER)**

This register is used to enable falling-edge triggered events on the corresponding interrupt pads.



**Figure 177. Interrupt Falling-Edge Event Enable Register (IFEER)**

1. 20 enabling events in 100-pin LQFP: IFEE[23:16] plus IFEE[11:0] (register bits 16–19 reserved).

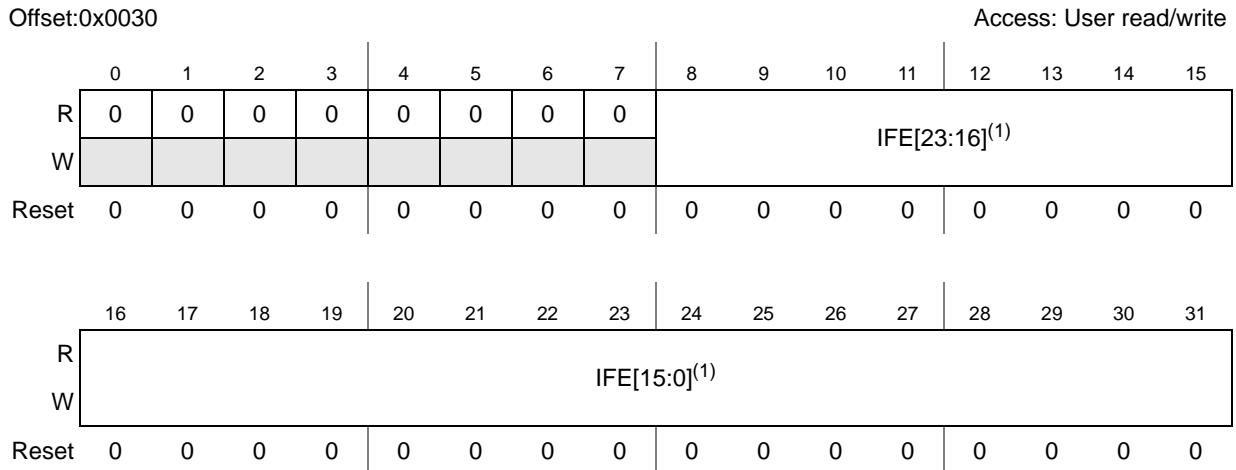
**Table 177. IFEER field descriptions**

Field	Description
IFEE[x]	Enable falling-edge events to cause the ISR(EIF[x]) bit to be set. 0 Falling-edge event is disabled 1 Falling-edge event is enabled

*Note: If both the IREER[IREE] and IFEER[IFEE] bits are cleared for the same interrupt source, the interrupt status flag for the corresponding external interrupt will never be set. If IREER[IREE] and IFEER[IFEE] bits are set for the same source the interrupts are triggered by both rising edge events and falling edge events.*

### 21.5.3.7 Interrupt Filter Enable Register (IFER)

This register is used to enable a digital filter counter on the corresponding interrupt pads to filter out glitches on the inputs.



**Figure 178. Interrupt Filter Enable Register (IFER)**

1. 20 bits in 100-pin LQFP: IFE[23:16] plus IFE[11:0] (register bits 16–19 reserved).

**Table 178. IFER field descriptions**

Field	Description
IFE[x]	Enable digital glitch filter on the interrupt pad input 0 Filter is disabled 1 Filter is enabled See the IFMC field descriptions in <a href="#">Table 189</a> for details on how the filter works.

### 21.5.3.8 Pad Configuration Registers (PCR0–PCR148)

The Pad Configuration Registers allow configuration of the static electrical and functional characteristics associated with I/O pads. Each PCR controls the characteristics of a single pad.

Please note that input and output peripheral muxing are separate.

- For output pads:
  - Select the appropriate alternate function in Pad Config Register (PCR)
  - OBE is not required for functions other than GPIO
- For INPUT pads:
  - Select the feature location from PSMI register
  - Set the IBE bit in the appropriate PCR
- For normal GPIO (not alternate function):
  - Configure PCR
  - Read from GPDI or write to GPDO

Offsets: Base + 0x0040 (PCR0)(149 registers)

Base + 0x0042 (PCR1)

Access: User read/write

...

Base + 0x0168 (PCR148)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SMC	APC		PA[1:0]	OBE	IBE		0	0	ODE	0	0	SRC	WPE	WPS
W																
Reset	0	0 <sup>(1)</sup>	0	0	0 <sup>(1)</sup>	0	0 <sup>(2)</sup>	0 <sup>(3)</sup>	0	0	0	0	0	0	0 <sup>(3)</sup>	1 <sup>(4)</sup>

Figure 179. Pad Configuration Registers (PCR<sub>x</sub>)

1. SMC and PA[1] are '1' for JTAG pads
2. OBE is '1' for TDO
3. IBE and WPE are '1' for TCK, TMS, TDI, FAB and ABS
4. WPS is '0' for input only pad with analog feature and FAB

Note: 16/32-bit access is supported.

In addition to the bit map above, the following [Table 180](#) describes the PCR depending on the pad type (pad types are defined in the “Pad types” section of this reference manual). The bits in shaded fields are not implemented for the particular I/O type. The PA field selecting the number of alternate functions may or may not be present depending on the number of alternate functions actually mapped on the pad.

Table 179. PCR bit implementation by pad type

Pad type	PCR bit No.															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S, M, F (Pad with GPIO and digital alternate function)		SMC	APC		PA[1:0]	OBE	IBE				ODE			SRC	WPE	WPS
Pad with slew rate control		SMC	APC		PA[1:0]	OBE	IBE				ODE			SRC	WPE	WPS
J (Pad with GPIO and analog functionality)		SMC	APC		PA[1:0]	OBE	IBE				ODE			SRC	WPE	WPS
I (Pad dedicated to ADC)		SMC	APC		PA[1:0]	OBE	IBE				ODE			SRC	WPE	WPS

Table 180. PCRx field descriptions

Field	Description
SMC	<p>Safe Mode Control.</p> <p>This bit supports the overriding of the automatic deactivation of the output buffer of the associated pad upon entering SAFE mode of the device.</p> <p>0 In SAFE mode, the output buffer of the pad is disabled.</p> <p>1 In SAFE mode, the output buffer remains functional.</p>
APC	<p>Analog Pad Control.</p> <p>This bit enables the usage of the pad as analog input.</p> <p>0 Analog input path from the pad is gated and cannot be used</p> <p>1 Analog input path switch can be enabled by the ADC</p>
PA[1:0]	<p>Pad Output Assignment</p> <p>This field is used to select the function that is allowed to drive the output of a multiplexed pad.</p> <p>00 Alternative Mode 0 — GPIO</p> <p>01 Alternative Mode 1 — See the signal description chapter</p> <p>10 Alternative Mode 2 — See the signal description chapter</p> <p>11 Alternative Mode 3 — See the signal description chapter</p> <p><b>Note:</b> Number of bits depends on the actual number of actual alternate functions. Please see datasheet.</p>
OBE	<p>Output Buffer Enable</p> <p>This bit enables the output buffer of the pad in case the pad is in GPIO mode.</p> <p>0 Output buffer of the pad is disabled when PA[1:0] = 00</p> <p>1 Output buffer of the pad is enabled when PA[1:0] = 00</p>
IBE	<p>Input Buffer Enable</p> <p>This bit enables the input buffer of the pad.</p> <p>0 Input buffer of the pad is disabled</p> <p>1 Input buffer of the pad is enabled</p>
ODE	<p>Open Drain Output Enable</p> <p>This bit controls output driver configuration for the pads connected to this signal. Either open drain or push/pull driver configurations can be selected. This feature applies to output pads only.</p> <p>0 Pad configured for push/pull output</p> <p>1 Pad configured for open drain</p>
SRC	<p>Slew Rate Control</p> <p>This field controls the slew rate of the associated pad when it is slew rate selectable. Its usage is the following:</p> <p>0 Pad configured as slow (default)</p> <p>1 Pad is configured as medium or fast (depending on the pad)</p> <p><b>Note:</b> PC[1] (TDO pad) is medium only. By default SRC = 0, and writing '1' has no effect.</p>
WPE	<p>Weak Pull Up/Down Enable</p> <p>This bit controls whether the weak pull up/down devices are enabled/disabled for the pad connected to this signal.</p> <p>0 Weak pull device disabled for the pad</p> <p>1 Weak pull device enabled for the pad</p>



**Table 180. PCRx field descriptions (continued)**

Field	Description
WPS	Weak Pull Up/Down Select This bit controls whether weak pull up or weak pull down devices are used for the pads connected to this signal when weak pull up/down devices are enabled. 0 Weak pull-down selected 1 Weak pull-up selected

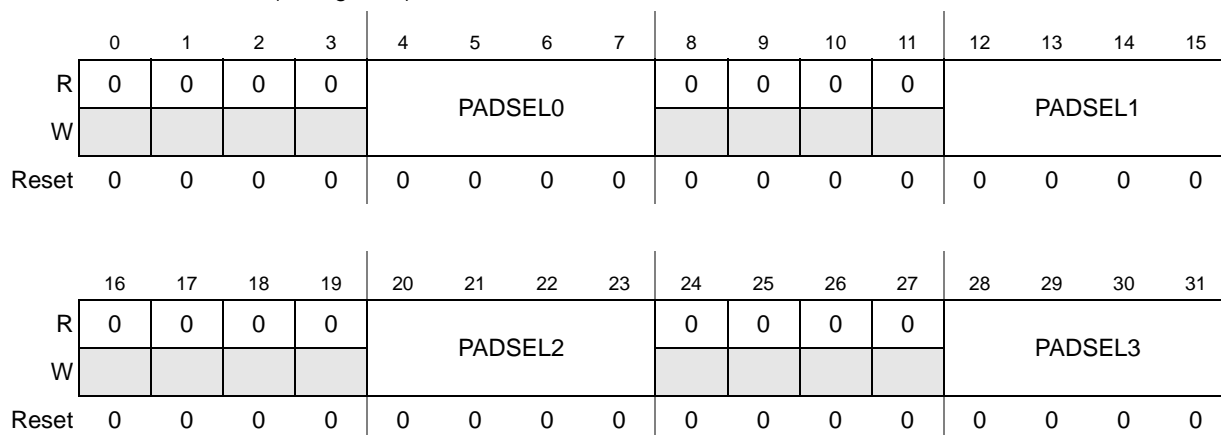
**21.5.3.9 Pad Selection for Multiplexed Inputs Registers (PSMI0\_3–PSMI60\_63)**

In some cases, a peripheral input signal can be selected from more than one pin. For example, the CAN1\_RXD signal can be selected on three different pins: PC[3], PC[11] and PF[15]. Only one can be active at a time. To select the pad to be used as input to the peripheral:

- Select the signal via the pad’s PCR register using the PA field.
- Specify the pad to be used via the appropriate PSMI field.

Offsets:0x0500–0x053C (16 registers)

Access: User read/write



**Figure 180. Pad Selection for Multiplexed Inputs Register (PSMI0\_3)**

**Table 181. PSMI0\_3 field descriptions**

Field	Description
PADSEL0–3, PADSEL4–7, ... PADSEL60–63	Pad Selection Bits Each PADSEL field selects the pad currently used for a certain input function. See <a href="#">Table 182</a> .

In order to multiplex different pads to the same peripheral input, the SIUL provides a register that controls the selection between the different sources.

Table 182. Peripheral input pin selection

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping <sup>(1)</sup>
PSMI0_3	PADSEL0	0x500	CAN1RX / FlexCAN_1	00: PCR[35] 01: PCR[43] 10: PCR[95] <sup>(2)</sup>
	PADSEL1	0x501	CAN2RX / FlexCAN_2	00: PCR[73] 01: PCR[89] <sup>(2)</sup>
	PADSEL2	0x502	CAN3RX / FlexCAN_3	00: PCR[36] 01: PCR[73] 10: PCR[89] <sup>(2)</sup>
	PADSEL3	0x503	CAN4RX / FlexCAN_4	00: PCR[35] 01: PCR[43] 10: PCR[95] <sup>(2)</sup>
PSMI4_7	PADSEL4	0x504	CAN5RX / FlexCAN_5	00: PCR[64] 01: PCR[97] <sup>(2)</sup>
	PADSEL5	0x505	SCK_0 / DSPI_0	00: PCR[14] 01: PCR[15]
	PADSEL6	0x506	CS0_0 / DSPI_0	00: PCR[14] 01: PCR[15] 10: PCR[27] <sup>(3)</sup>
	PADSEL7	0x507	SCK_1 / DSPI_1	00: PCR[34] 01: PCR[68] 10: PCR[114] <sup>(2)</sup>
PSMI8_11	PADSEL8	0x508	SIN_1 / DSPI_1	00: PCR[36] 01: PCR[66] 10: PCR[112] <sup>(2)</sup>
	PADSEL9	0x509	CS0_1 / DSPI_1	00: PCR[] 01: PCR[61] 10: PCR[69] 11: PCR[115] <sup>(2)</sup>
	PADSEL10	0x50A	SCK_2 / DSPI_2	00: PCR[46] 01: PCR[78] <sup>(2)</sup> 10: PCR[105] <sup>(2)</sup>
	PADSEL11	0x50B	SIN_2 / DSPI_2	00: PCR[44] 01: PCR[76]
PSMI12_15	PADSEL12	0x50C	CS0_2 / DSPI_2	00: PCR[47] 01: PCR[79] <sup>(2)</sup> 10: PCR[82] <sup>(2)</sup> 11: PCR[104] <sup>(2)</sup>
	PADSEL13	0x50D	E1UC[3] / eMIOS_0	00: PCR[3] 01: PCR[27] <sup>(3)</sup> 10: PCR[40]
	PADSEL14	0x50E	E0UC[4] / eMIOS_0	00: PCR[4] 01: PCR[28]
	PADSEL15	0x50F	E0UC[5] / eMIOS_0	00: PCR[5] 01: PCR[29]



Table 182. Peripheral input pin selection (continued)

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping <sup>(1)</sup>
PSMI16_19	PADSEL16	0x510	E0UC[6] / eMIOS_0	00: PCR[6] 01: PCR[30]
	PADSEL17	0x511	E0UC[7] / eMIOS_0	00: PCR[7] 01: PCR[31] 10: PCR[41]
	PADSEL18	0x512	E0UC[10] / eMIOS_0	00: PCR[10] 01: PCR[80] <sup>(2)</sup>
	PADSEL19	0x513	E0UC[11] / eMIOS_0	00: PCR[11] 01: PCR[81] <sup>(2)</sup>
PSMI20_23	PADSEL20	0x514	E0UC[12] / eMIOS_0	00: PCR[44] 01: PCR[82] <sup>(2)</sup>
	PADSEL21	0x515	E0UC[13] / eMIOS_0	00: PCR[45] 01: PCR[83] <sup>(2)</sup> 10: PCR[0]
	PADSEL22	0x516	E0UC[14] / eMIOS_0	00: PCR[46] 01: PCR[84] <sup>(2)</sup> 10: PCR[8]
	PADSEL23	0x517	E0UC[22] / eMIOS_0	00: PCR[70] 01: PCR[72] 10: PCR[85] <sup>(2)</sup>
PSMI24_27	PADSEL24	0x518	E0UC[23] / eMIOS_0	00: PCR[71] 01: PCR[73] 10: PCR[86] <sup>(2)</sup>
	PADSEL25	0x519	E0UC[24] / eMIOS_0	00: PCR[60] 01: PCR[106] <sup>(2)</sup> 10: PCR[75]
	PADSEL26	0x51A	E0UC[25] / eMIOS_0	00: PCR[61] 01: PCR[107] <sup>(2)</sup>
	PADSEL27	0x51B	E0UC[26] / eMIOS_0	00: PCR[62] 01: PCR[108] <sup>(2)</sup>
PSMI28_31	PADSEL28	0x51C	E0UC[27] / eMIOS_0	00: PCR[63] 01: PCR[109] <sup>(2)</sup>
	PADSEL29	0x51D	SCL / f_0	00: PCR[11] 01: PCR[19]
	PADSEL30	0x51E	SDA / I2C__0	00: PCR[10] 01: PCR[18]
	PADSEL31	0x51F	LIN3RX / LINFlex_3	00: PCR[8] 01: PCR[75]

Table 182. Peripheral input pin selection (continued)

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping <sup>(1)</sup>
PSMI32_35	PADSEL32	0x520	SCK_3 / DSPI_3	00: PCR[100] <sup>(2)</sup> 01: PCR[124] <sup>(3)</sup>
	PADSEL33	0x521	SIN_3 / DSPI_3	00: PCR[101] <sup>(2)</sup> 01: PCR[139] <sup>(3)</sup>
	PADSEL34	0x522	CS0_3 / DSPI_3	00: PCR[99] <sup>(2)</sup> 01: PCR[125] <sup>(3)</sup> 10: PCR[140] <sup>(3)</sup>
	PADSEL35	0x523	SCK_4 / DSPI_4	00: PCR[109] <sup>(2)</sup> 01: PCR[126] <sup>(3)</sup> 10: PCR[133] <sup>(3)</sup>
PSMI36_39	PADSEL36	0x524	SIN_4 / DSPI_4Reserved	00: PCR[106] <sup>(2)</sup> 01: PCR[142] <sup>(3)</sup> —
	PADSEL37	0x525	CS0_4 / DSPI_4Reserved	00: PCR[107] <sup>(2)</sup> 01: PCR[123] <sup>(3)</sup> 10: PCR[134] <sup>(3)</sup> 11: PCR[143] <sup>(3)</sup> —
	PADSEL38	0x526	E0UC[0] / eMIOS_0	00: PCR[0] 01: PCR[14]
	PADSEL39	0x527	E0UC[1] / eMIOS_0	00: PCR[1] 01: PCR[15]
PSMI40_43	PADSEL40	0x528	E0UC[28] / eMIOS_0	00: PCR[12] 01: PCR[128] <sup>(3)</sup>
	PADSEL41	0x529	E0UC[29] / eMIOS_0	00: PCR[13] 01: PCR[129] <sup>(3)</sup>
	PADSEL42	0x52A	E0UC[30] / eMIOS_0	00: PCR[16] 01: PCR[18] 10: PCR[130] <sup>(3)</sup>
	PADSEL43	0x52B	E0UC[31] / eMIOS0	00: PCR[17] 01: PCR[19] 10: PCR[131] <sup>(3)</sup>
PSMI44_47	PADSEL44	0x52C	E1UC[1] / eMIOS_1	00: PCR[111] <sup>(2)</sup> 01: PCR[89] <sup>(2)</sup>
	PADSEL45	0x52D	E1UC[2] / eMIOS_1	00: PCR[112] <sup>(2)</sup> 01: PCR[90] <sup>(2)</sup>
	PADSEL46	0x52E	E1UC[3] / eMIOS_1	00: PCR[113] <sup>(2)</sup> 01: PCR[91] <sup>(2)</sup>
	PADSEL47	0x52F	E1UC[4] / eMIOS_1	00: PCR[114] <sup>(2)</sup> 01: PCR[95] <sup>(2)</sup>

**Table 182. Peripheral input pin selection (continued)**

PSMI registers	PADSEL fields	SIUL address offset	Function / Peripheral	Mapping <sup>(1)</sup>
PSMI48_51	PADSEL48	0x530	E1UC[5] / eMIOS_1	00: PCR[115] <sup>(2)</sup> 01: PCR[123] <sup>(3)</sup>
	PADSEL49	0x531	E1UC[17] / eMIOS_1	00: PCR[104] <sup>(2)</sup> 01: PCR[127] <sup>(3)</sup>
	PADSEL50	0x532	E1UC[18] / eMIOS_1	00: PCR[105] <sup>(2)</sup> 01: PCR[148]
	PADSEL51	0x533	E1UC[25] / eMIOS_1	00: PCR[92] <sup>(2)</sup> 01: PCR[124] <sup>(3)</sup>
PSMI52_55	PADSEL52	0x534	E1UC[26] / eMIOS_1	00: PCR[93] <sup>(2)</sup> 01: PCR[125] <sup>(3)</sup>
	PADSEL53	0x535	E1UC[27] / eMIOS_1	00: PCR[94] <sup>(2)</sup> 01: PCR[126] <sup>(3)</sup>
	PADSEL54	0x536	E1UC[28] / eMIOS_1	00: PCR[38] 01: PCR[132] <sup>(3)</sup>
	PADSEL55	0x537	E1UC[29] / eMIOS_1	00: PCR[39] 01: PCR[133] <sup>(3)</sup>
PSMI56_59	PADSEL56	0x538	E1UC[30] / eMIOS_1Reserved	00: PCR[74] 01: PCR[103] <sup>(2)</sup> 10: PCR[134] <sup>(3)</sup> —
	PADSEL57	0x539	E1UC[31] / eMIOS_1Reserved	00: PCR[36] 01: PCR[106] <sup>(2)</sup> 10: PCR[135] <sup>(3)</sup> —
	PADSEL58	0x53A	LIN2RX / LINFlex _2	00: PCR[41] 01: PCR[11]
	PADSEL59	0x53B	LIN4RX / LINFlex _4Reserved	00: PCR[6] 01: PCR[91] <sup>(2)</sup> —
PSMI60_63 <sup>(4)</sup>	PADSEL60	0x53C	LIN5RX / LINFlex _5	00: PCR[4] 01: PCR[93] <sup>(2)</sup>
	PADSEL61	0x53D	LIN8RX / LINFlex _8	00: PCR[111] <sup>(2)</sup> 01: PCR[129] <sup>(3)</sup>
	PADSEL62	0x53E	LIN0RX/LINFlexD_0	00: PCR[19] 01: PCR[17]

1. See the signal description chapter of this reference manual for correspondence between PCR and pinout
2. Not available in 100-pin LQFP
3. Available only in 176-pin LQFP and 208 BGA packages
4. PADSEL63 not implemented

**21.5.3.10 GPIO Pad Data Output Registers (GPDO0\_3–GPDO148\_151)**

These registers are used to set or clear GPIO pads. Each pad data out bit can be controlled separately with a byte access.



Offsets: 0x0600–0x06A0 (38 registers)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO[0]	0	0	0	0	0	0	0	PDO[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO[2]	0	0	0	0	0	0	0	PDO[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 181. Port GPIO Pad Data Output Register 0–3 (GPDO0\_3)

Table 183. GPDO0\_3 field descriptions

Field	Description
PDO[x]	Pad Data Out This bit stores the data to be driven out on the external GPIO pad controlled by this register. 0 Logic low value is driven on the corresponding GPIO pad when the pad is configured as an output 1 Logic high value is driven on the corresponding GPIO pad when the pad is configured as an output

**Caution:** Toggling several IOs at the same time can significantly increase the current in a pad group. Caution must be taken to avoid exceeding maximum current thresholds. Please see datasheet.

**21.5.3.11 GPIO Pad Data Input Registers (GPDIO\_3–GPD1148\_151)**

These registers are used to read the GPIO pad data with a byte access.

Offsets: 0x0800–0x08A0 (38 registers)

Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI[0]	0	0	0	0	0	0	0	PDI[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI[2]	0	0	0	0	0	0	0	PDI[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 182. Port GPIO Pad Data Input Register 0–3 (GPDIO\_3)

Table 184. GPDIO\_3 field descriptions

Field	Description
PDI[x]	Pad Data In This bit stores the value of the external GPIO pad associated with this register. 0 Value of the data in signal for the corresponding GPIO pad is logic low 1 Value of the data in signal for the corresponding GPIO pad is logic high

### 21.5.3.12 Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO4)

SPC560B54/6x devices ports are constructed such that they contain 16 GPIO pins, for example PortA[0..15]. Parallel port registers for input (PGPDI) and output (PGPDO) are provided to allow a complete port to be written or read in one operation, dependent on the individual pad configuration.

Writing a parallel PGPDO register directly sets the associated GPDO register bits. There is also a masked parallel port output register allowing the user to determine which pins within a port are written.

While very convenient and fast, this approach does have implications regarding current consumption for the device power segment containing the port GPIO pads. Toggling several GPIO pins simultaneously can significantly increase current consumption.

**Caution:** Caution must be taken to avoid exceeding maximum current thresholds when toggling multiple GPIO pins simultaneously. Please see datasheet.

[Table 185](#) shows the locations and structure of the PGPDOx registers.

**Table 185. PGPDO0 – PGPDO4 register map**

Offset (1)	Register	Field																															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x0C00	PGPDO0	Port A																Port B															
0x0C04	PGPDO1	Port C																Port D															
0x0C08	PGPDO2	Port E																Port F															
0x0C0C	PGPDO3	Port G																Port H															
0x0C10	PGPDO4	Port I																Port J				Reserved											

1. SIU base address is 0xC3F9\_0000. To calculate register address add offset to base address

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in [Table 185](#), the PGPDO0 register contains fields for Port A and Port B.

- Bit 0 is mapped to Port A[0], bit 1 is mapped to Port A[1] and so on, through bit 15, which is mapped to Port A[15]
- Bit 16 is mapped to Port B[0], bit 17 is mapped to Port B[1] and so on, through bit 31, which is mapped to Port B[15].

**21.5.3.13 Parallel GPIO Pad Data In Registers (PGPDI0 – PGPDI4)**

The SIU\_PGPDI registers are similar in operation to the PGPDIO registers, described in the previous section ([Section 21.5.3.12: Parallel GPIO Pad Data Out Registers \(PGPDO0 – PGPDO4\)](#)) but they are used to read port pins simultaneously.

*Note:* The port pins to be read need to be configured as inputs but even if a single pin within a port has IBE set, then you can still read that pin using the parallel port register. However, this does mean you need to be very careful.

Reads of PGPDI registers are equivalent to reading the corresponding GPDIO registers but significantly faster since as many as two ports can be read simultaneously with a single 32-bit read operation.

[Table 186](#) shows the locations and structure of the PGPDIx registers. Each 32-bit PGPDIx register contains two 16-bit fields, each field containing the values for a separate port.

**Table 186. PGPDI0 – PGPDI4 register map**

Offset (1)	Register	Field																															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x0C40	PGPDI0	Port A																Port B															
0x0C44	PGPDI1	Port C																Port D															
0x0C48	PGPDI2	Port E																Port F															
0x0C4C	PGPDI3	Port G																Port H															
0x0C50	PGPDI4	Port I																Port J				Reserved											

1. SIU base address is 0xC3F9\_0000. To calculate register address add offset to base address.





It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in [Table 186](#), the PGPDI0 register contains fields for Port A and Port B.

- Bit 0 is mapped to Port A[0], bit 1 is mapped to Port A[1] and so on, through bit 15, which is mapped to Port A[15]
- Bit 16 is mapped to Port B[0], bit 17 is mapped to Port B[1] and so on, through bit 31, which is mapped to Port B[15].

### 21.5.3.14 Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO9)

The MPGPDOx registers are similar in operation to the PGPDOx ports described in [Section 21.5.3.12: Parallel GPIO Pad Data Out Registers \(PGPDO0 – PGPDO4\)](#), but with two significant differences:

- The MPGPDOx registers support *masked* port-wide changes to the data out on the pads of the respective port. Masking effectively allows selective bitwise writes to the full 16-bit port.
- Each 32-bit MPGPDOx register is associated to only one port.

*Note:* The MPGPDOx registers may only be accessed with 32-bit writes. 8-bit or 16-bit writes will not modify any bits in the register and will cause a transfer error response by the module. Read accesses return '0'.

[Table 187](#) shows the locations and structure of the MPGPDOx registers. Each 32-bit MPGPDOx register contains two 16-bit fields (MASK<sub>x</sub> and MPPDO<sub>x</sub>). The MASK field is a bitwise mask for its associated port. The MPPDO0 field contains the data to be written to the port.

**Table 187. MPGPDO0 – MPGPDO9 register map**

Offset <sup>(1)</sup>	Register	Field																															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x0C80	MPGPDO0	MASK0 (Port A)																MPPDO0 (Port A)															
0x0C84	MPGPDO1	MASK1 (Port B)																MPPDO1 (Port B)															
0x0C88	MPGPDO2	MASK2 (Port C)																MPPDO2 (Port C)															
0x0C8C	MPGPDO3	MASK3 (Port D)																MPPDO3 (Port D)															
0x0C90	MPGPDO4	MASK4 (Port E)																MPPDO4 (Port E)															
0x0C94	MPGPDO5	MASK5 (Port F)																MPPDO5 (Port F)															
0x0C98	MPGPDO6	MASK6 (Port G)																MPPDO6 (Port G)															
0x0C9C	MPGPDO7	MASK7 (Port H)																MPPDO7 (Port H)															
0x0CA0	MPGPDO8	MASK8 (Port I)																MPPDO8 (Port I)															
0x0CAF	MPGPDO9	MASK9 (Port J)				Reserved												MPPDO9 (Port J)				Reserved											

1. SIU base address is 0xC3F9\_0000. To calculate register address add offset to base address.

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.



For example in [Table 187](#), the MPGPDO0 register contains field MASK0, which is the bitwise mask for Port A and field MPPDO0, which contains data to be written to Port A.

- MPGPDO0[0] is the mask bit for Port A[0], MPGPDO0[1] is the mask bit for Port A[1] and so on, through MPGPDO0[15], which is the mask bit for Port A[15]
- MPGPDO0[16] is the data bit mapped to Port A[0], MPGPDO0[17] is mapped to Port A[1] and so on, through MPGPDO0[31], which is mapped to Port A[15].

**Table 188. MPGPDO0..MPGPDO9 field descriptions**

Field	Description
MASK <sub>x</sub> [15:0]	Mask Field Each bit corresponds to one data bit in the MPPDO <sub>x</sub> register at the same bit location. 0 Associated bit value in the MPPDO <sub>x</sub> field is ignored 1 Associated bit value in the MPPDO <sub>x</sub> field is written
MPPDO <sub>x</sub> [15:0]	Masked Parallel Pad Data Out Write the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bitwise GPIO Pad Data Output Registers (GPDO0_3–GPDO148_151). The x and bit index define which MPPDO register bit is equivalent to which PDO register bit according to the following equation: MPPDO[x][y] = PDO[(x*16)+y]

**Caution:** Toggling several IOs at the same time can significantly increase the current in a pad group. Caution must be taken to avoid exceeding maximum current thresholds. Please see datasheet.

**21.5.3.15 Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)**

These registers are used to configure the filter counter associated with each digital glitch filter.

*Note:* For the pad transition to trigger an interrupt it must be steady for at least the filter period.

Offset: 0x1000–0x105C) (24 registers)

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	MAXCNTx			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 183. Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)**



**Table 189. IFMC field descriptions**

Field	Description
MAXCNTx	Maximum Interrupt Filter Counter setting $\text{Filter Period} = T(\text{CK}) * \text{MAXCNTx} + n * T(\text{CK})$ Where (n can be -1 to 3) MAXCNTx can be 0 to 15 T(CK): Prescaled Filter Clock Period, which is FIRC clock prescaled to IFCP value T(FIRC): Basic Filter Clock Period: 62.5 ns ( $f_{\text{FIRC}} = 16 \text{ MHz}$ )

**21.5.3.16 Interrupt Filter Clock Prescaler Register (IFCPR)**

This register is used to configure a clock prescaler which is used to select the clock for all digital filter counters in the SIUL.

Offsets: 0x1080

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	IFCP			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 184. Interrupt Filter Clock Prescaler Register (IFCPR)**

**Table 190. IFCPR field descriptions**

Field	Description
IFCP	Interrupt Filter Clock Prescaler setting $\text{Prescaled Filter Clock Period} = T(\text{FIRC}) \times (\text{IFCP} + 1)$ T(FIRC) is the fast internal RC oscillator period. IFCP can be 0 to 15.

**21.6 Functional description**

**21.6.1 Pad control**

The SIUL controls the configuration and electrical characteristic of the device pads. It provides a consistent interface for all pads, both on a by-port and a by-bit basis. The pad configuration registers (PCRn, see [Section 21.5.3.8: Pad Configuration Registers \(PCR0–](#)

[PCR148](#)) allow software control of the static electrical characteristics of external pins with a single write. These are used to configure the following pad features:

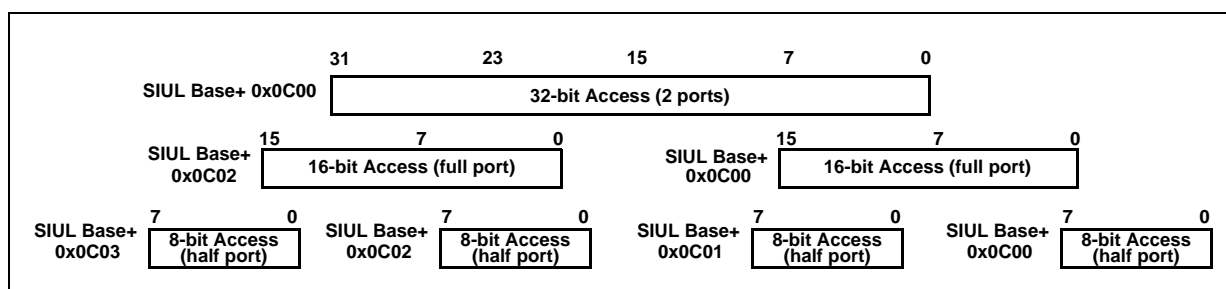
- Open drain output enable
- Slew rate control
- Pull control
- Pad assignment
- Control of analog path switches
- Safe mode behavior configuration

### 21.6.2 General purpose input and output pads (GPIO)

The SIUL manages up to 149 GPIO pads organized as ports that can be accessed for data reads and writes as 32, 16 or 8-bit<sup>(r)</sup>.

*Note:* Ports are organized as groups of 16 GPIO pads, with the exception of Port J, which has 5. A 32-bit R/W operation accesses two ports simultaneously. A 16-bit operation accesses a full port and an 8-bit access either the upper or lower byte of a port.

As shown in [Figure 185](#), all port accesses are identical with each read or write being performed only at a different location to access a different port width.



**Figure 185. Data Port example arrangement showing configuration for different port width accesses**

The SIUL has separate data input (GPDIn<sub>n</sub>, see [Section 21.5.3.11: GPIO Pad Data Input Registers \(GPDIO\\_3–GPDIO148\\_151\)](#)) and data output (GPDO<sub>n</sub>, see [Section 21.5.3.10: GPIO Pad Data Output Registers \(GPDOO\\_3–GPDO148\\_151\)](#)) registers for all pads, allowing the possibility of reading back an input or output value of a pad directly. This supports the ability to validate what is present on the pad rather than simply confirming the value that was written to the data register by accessing the data input registers.

Data output registers allow an output pad to be driven high or low (with the option of push-pull or open drain drive). Input registers are read-only and reflect the respective pad value.

When the pad is configured to use one of its alternate functions, the data input value reflects the respective value of the pad. If a write operation is performed to the data output register for a pad configured as an alternate function (non-GPIO), this write will not be reflected by the pad value until reconfigured to GPIO.

<sup>r</sup> There are exceptions. Some pads, e.g., precision analog pads, are input only.

The allocation of what input function is connected to the pin is defined by the PSMI registers (PCR<sub>n</sub>, see [Section 21.5.3.9: Pad Selection for Multiplexed Inputs Registers \(PSMIO\\_3–PSMI60\\_63\)](#)).

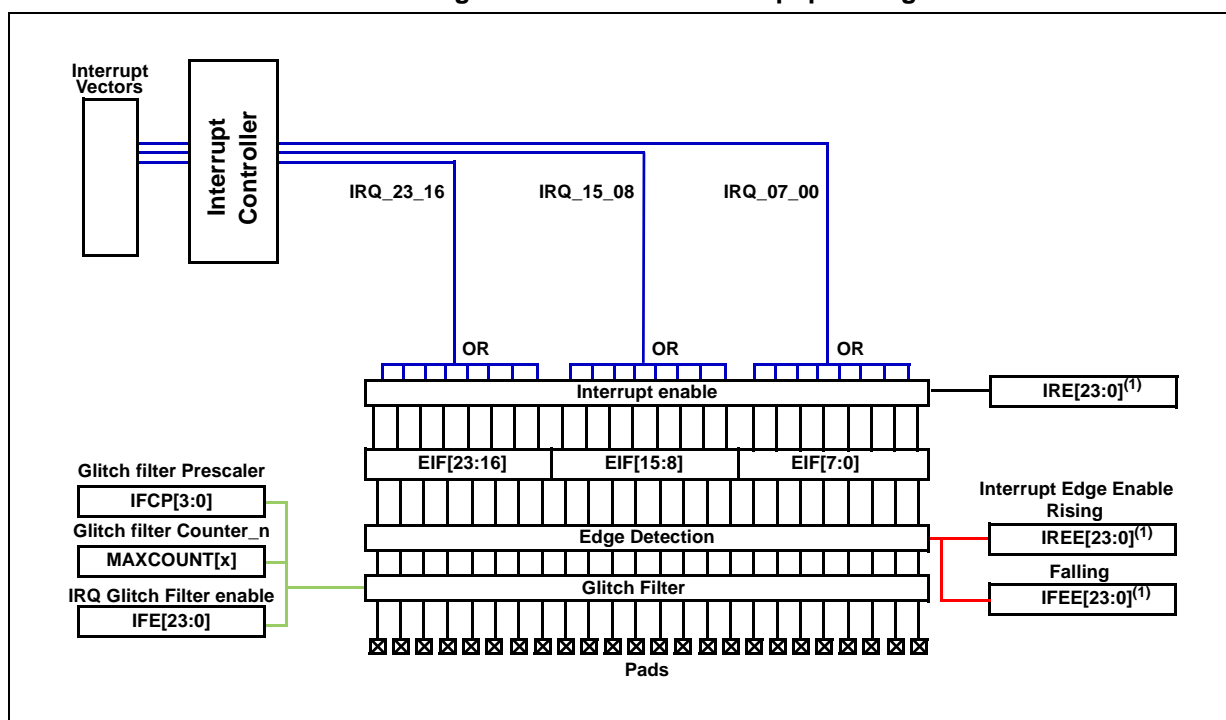
### 21.6.3 External interrupts

The SIUL supports 24 external interrupts, EIRQ0–EIRQ23. In the signal description chapter of this reference manual, mapping is shown for external interrupts to pads.

The SIUL supports three interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts combined together with the presence of flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

See [Figure 186](#) for an overview of the external interrupt implementation.

**Figure 186. External interrupt pad diagram**



1. 20 interrupts in 100-pin LQFP.

Each interrupt can be enabled or disabled independently. This can be performed using the IREER. A pad defined as an external interrupt can be configured to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled is reserved and should not be configured.

The active EIRQ edge is controlled through the configuration of the registers IREER and IFEER.

Each external interrupt supports an individual flag which is held in the Interrupt Status Flag Register (ISR). The bits in the ISR[EIF] field are cleared by writing a '1' to them; this prevents inadvertent overwriting of other flags in the register.

## 21.7 Pin muxing

For pin muxing, please see the signal description chapter of this reference manual.

## 22 Inter-Integrated Circuit Bus Controller Module (I<sup>2</sup>C)

### 22.1 Introduction

#### 22.1.1 Overview

The Inter-Integrated Circuit (I<sup>2</sup>C™ or IIC) bus is a two wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the number of external connections to devices and does not require an external address decoder.

This bus is suitable for applications requiring occasional communications over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface is designed to operate up to 100 kbps in Standard Mode and 400 Kbps in Fast Mode. The device is capable of operating at higher baud rates, up to a maximum of module clock/20 with reduced bus loading. Actual baud rate can be less than the programmed baud rate and is dependent on the SCL rise time. SCL rise time is dependent on the external pullup resistor value and bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

#### 22.1.2 Features

The I<sup>2</sup>C module has the following key features:

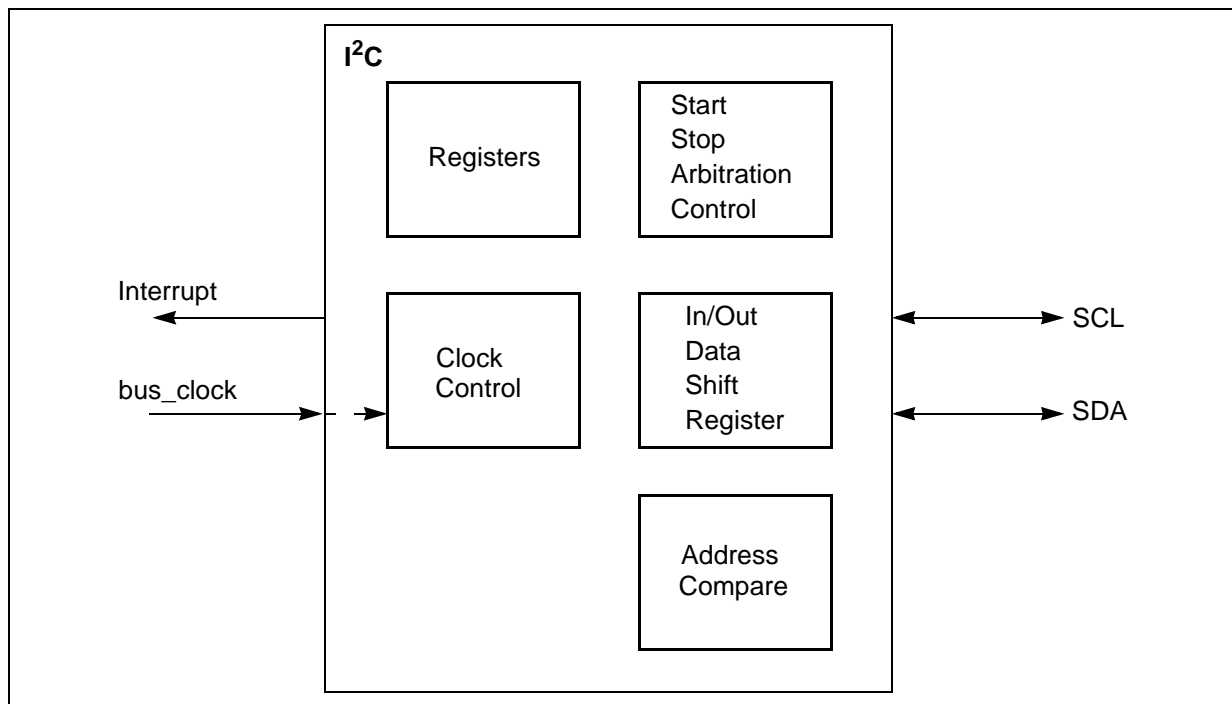
- Compatible with I<sup>2</sup>C Bus standard
- Multi-master operation
- Software programmable for one of 256 different serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection

Features currently not supported:

- No support for general call address
- Not compliant to ten-bit addressing

#### 22.1.3 Block diagram

The block diagram of the I<sup>2</sup>C module is shown in [Figure 187](#).

Figure 187. I<sup>2</sup>C block diagram

## 22.2 External signal description

The Inter-Integrated Circuit (I<sup>2</sup>C) module has two external pins, SCL and SDA.

### 22.2.1 SCL

This is the bidirectional Serial Clock Line (SCL) of the module, compatible with the I<sup>2</sup>C-Bus specification.

### 22.2.2 SDA

This is the bidirectional Serial Data line (SDA) of the module, compatible with the I<sup>2</sup>C-Bus specification.

## 22.3 Memory map and register description

### 22.3.1 Module memory map

The memory map for the I<sup>2</sup>C module is given below in [Table 191](#). The total address for each register is the sum of the base address for the I<sup>2</sup>C module and the address offset for each register.



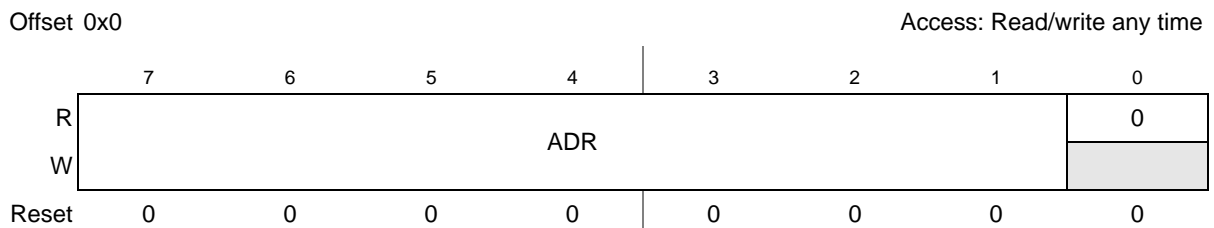
**Table 191. I<sup>2</sup>C memory map**

Base address: 0xFFE3_0000		
Address offset	Register	Location
0x0	I <sup>2</sup> C Bus Address Register (IBAD)	<a href="#">on page 441</a>
0x1	I <sup>2</sup> C Bus Frequency Divider Register (IBFD)	<a href="#">on page 442</a>
0x2	I <sup>2</sup> C Bus Control Register (IBCR)	<a href="#">on page 448</a>
0x3	I <sup>2</sup> C Bus Status Register (IBSR)	<a href="#">on page 449</a>
0x4	I <sup>2</sup> C Bus Data I/O Register (IBDR)	<a href="#">on page 450</a>
0x5	I <sup>2</sup> C Bus Interrupt Config Register (IBIC)	<a href="#">on page 451</a>

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the IBDF register for the frequency divider is accessible by a 16-bit read/write to address Base + 0x000, but performing a 16-bit access to Base + 0x001 is illegal.

### 22.3.2 I<sup>2</sup>C Bus Address Register (IBAD)

This register contains the address the I<sup>2</sup>C bus will respond to when addressed as a slave; note that it is not the address sent on the bus during the address transfer.

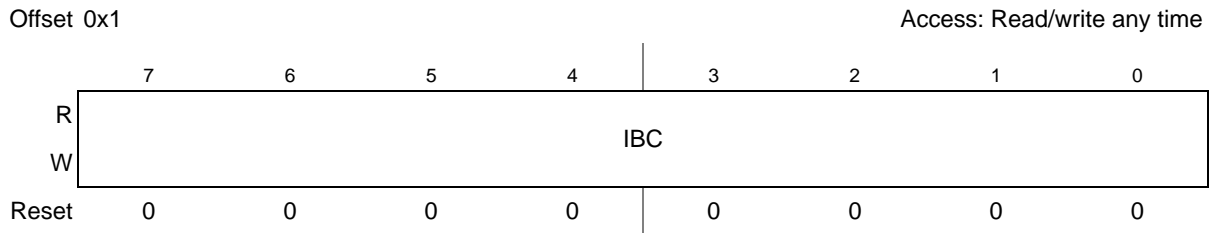


**Figure 188. I<sup>2</sup>C Bus Address Register (IBAD)**

**Table 192. IBAD field descriptions**

Field	Description
ADR	Slave Address. Specific slave address to be used by the I <sup>2</sup> C Bus module. <b>Note:</b> The default mode of I <sup>2</sup> C Bus is slave mode for an address match on the bus.

### 22.3.3 I<sup>2</sup>C Bus Frequency Divider Register (IBFD)



**Figure 189. I<sup>2</sup>C Bus Frequency Divider Register (IBFD)**

**Table 193. IBFD field descriptions**

Field	Description
IBC	I-Bus Clock Rate. This field is used to prescale the clock for bit rate selection. The bit clock generator is implemented as a prescale divider. The IBC bits are decoded to give the Tap and Prescale values as follows: 7–6 select the prescaled shift register (see <a href="#">Table 194</a> ) 5–3 select the prescaler divider (see <a href="#">Table 195</a> ) 2–0 select the shift register tap point (see <a href="#">Table 196</a> )

**Table 194. I-Bus multiplier factor**

IBC7–6	MUL
00	01
01	02
10	04
11	RESERVED

**Table 195. I-Bus prescaler divider values**

IBC5–3	scl2start (clocks)	scl2stop (clocks)	scl2tap (clocks)	tap2tap (clocks)
000	2	7	4	1
001	2	7	4	2
010	2	9	6	4
011	6	9	6	8
100	14	17	14	16
101	30	33	30	32
110	62	65	62	64
111	126	129	126	128

Table 196. I-Bus tap and prescale values

IBC2-0	SCL Tap (clocks)	SDA Tap (clocks)
000	5	1
001	6	1
010	7	2
011	8	2
100	9	3
101	10	3
110	12	4
111	15	4

The number of clocks from the falling edge of SCL to the first tap (Tap[1]) is defined by the values shown in the scl2tap column of [Table 195](#). All subsequent tap points are separated by  $2^{IBC5-3}$  as shown in the tap2tap column in [Table 195](#). The SCL Tap is used to generate the SCL period and the SDA Tap is used to determine the delay from the falling edge of SCL to the change of state of SDA i.e. the SDA hold time.

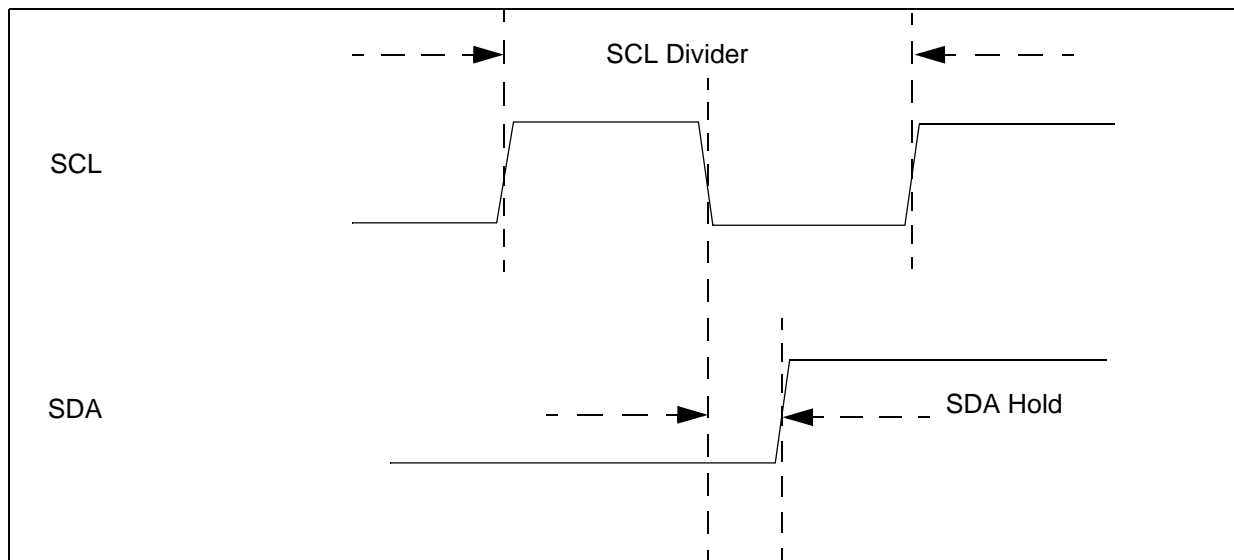
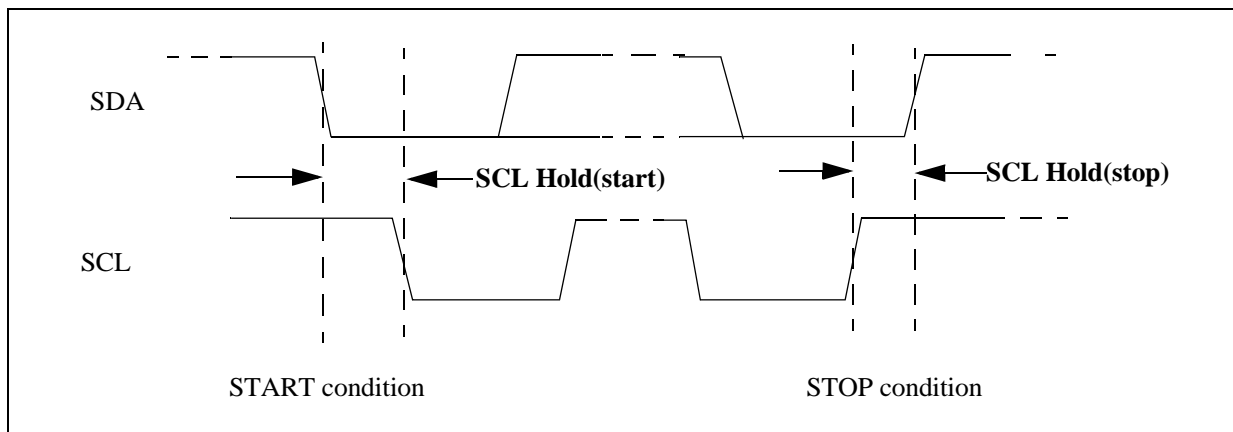


Figure 190. SDA hold time



**Figure 191. SCL divider and SDA hold**

The equation used to generate the divider values from the IBFD bits is:

**Equation 3** 
$$\text{SCL Divider} = \text{MUL} \times \{2 \times (\text{scl2tap} + [(\text{SCL\_Tap} - 1) \times \text{tap2tap}] + 2)\}$$

The SDA hold delay is equal to the CPU clock period multiplied by the SDA Hold value shown in [Table 197](#). The equation used to generate the SDA Hold value from the IBFD bits is:

**Equation 4** 
$$\text{SDA Hold} = \text{MUL} \times \{\text{scl2tap} + [(\text{SDA\_Tap} - 1) \times \text{tap2tap}] + 3\}$$

The equation for SCL Hold values to generate the start and stop conditions from the IBFD bits is:

**Equation 5** 
$$\text{SCL Hold(start)} = \text{MUL} \times [\text{scl2start} + (\text{SCL\_Tap} - 1) \times \text{tap2tap}]$$

**Equation 6** 
$$\text{SCL Hold(stop)} = \text{MUL} \times [\text{scl2stop} + (\text{SCL\_Tap} - 1) \times \text{tap2tap}]$$

Table 197. I<sup>2</sup>C divider and hold values

	IBC7-0 (hex)	SCL divider (clocks)	SDA hold (clocks)	SCL hold (start)	SCL hold (stop)
MUL = 1	00	20	7	6	11
	01	22	7	7	12
	02	24	8	8	13
	03	26	8	9	14
	04	28	9	10	15
	05	30	9	11	16
	06	34	10	13	18
	07	40	10	16	21
	08	28	7	10	15
	09	32	7	12	17
	0A	36	9	14	19
	0B	40	9	16	21
	0C	44	11	18	23
	0D	48	11	20	25
	0E	56	13	24	29
	0F	68	13	30	35
	10	48	9	18	25
	11	56	9	22	29
	12	64	13	26	33
	13	72	13	30	37
	14	80	17	34	41
	15	88	17	38	45
	16	104	21	46	53
	17	128	21	58	65
	18	80	9	38	41
	19	96	9	46	49
	1A	112	17	54	57
	1B	128	17	62	65
	1C	144	25	70	73
	1D	160	25	78	81
	1E	192	33	94	97
	1F	240	33	118	121
	20	160	17	78	81
	21	192	17	94	97
	22	224	33	110	113
	23	256	33	126	129
	24	288	49	142	145
	25	320	49	158	161
	26	384	65	190	193
	27	480	65	238	241
	28	320	33	158	161
	29	384	33	190	193
	2A	448	65	222	225
	2B	512	65	254	257
	2C	576	97	286	289
	2D	640	97	318	321
	2E	768	129	382	385
	2F	960	129	478	481
30	640	65	318	321	
31	768	65	382	385	
32	896	129	446	449	
33	1024	129	510	513	
34	1152	193	574	577	
35	1280	193	638	641	
36	1536	257	766	769	
37	1920	257	958	961	
38	1280	129	638	641	
39	1536	129	766	769	
3A	1792	257	894	897	
3B	2048	257	1022	1025	
3C	2304	385	1150	1153	
3D	2560	385	1278	1281	
3E	3072	513	1534	1537	
3F	3840	513	1918	1921	

Table 197. I<sup>2</sup>C divider and hold values (continued)

	IBC7-0 (hex)	SCL divider (clocks)	SDA hold (clocks)	SCL hold (start)	SCL hold (stop)
MUL = 2	40	40	14	12	22
	41	44	14	14	24
	42	48	16	16	26
	43	52	16	18	28
	44	56	18	20	30
	45	60	18	22	32
	46	68	20	26	36
	47	80	20	32	42
	48	56	14	20	30
	49	64	14	24	34
	4A	72	18	28	38
	4B	80	18	32	42
	4C	88	22	36	46
	4D	96	22	40	50
	4E	112	26	48	58
	4F	136	26	60	70
	50	96	18	36	50
	51	112	18	44	58
	52	128	26	52	66
	53	144	26	60	74
	54	160	34	68	82
	55	176	34	76	90
	56	208	42	92	106
	57	256	42	116	130
	58	160	18	76	82
	59	192	18	92	98
	5A	224	34	108	114
	5B	256	34	124	130
	5C	288	50	140	146
	5D	320	50	156	162
	5E	384	66	188	194
	5F	480	66	236	242
	60	320	28	156	162
	61	384	28	188	194
	62	448	32	220	226
	63	512	32	252	258
	64	576	36	284	290
	65	640	36	316	322
	66	768	40	380	386
	67	960	40	476	482
	68	640	28	316	322
	69	768	28	380	386
	6A	896	36	444	450
	6B	1024	36	508	514
	6C	1152	44	572	578
	6D	1280	44	636	642
	6E	1536	52	764	770
	6F	1920	52	956	962
70	1280	36	636	642	
71	1536	36	764	770	
72	1792	52	892	898	
73	2048	52	1020	1026	
74	2304	68	1148	1154	
75	2560	68	1276	1282	
76	3072	84	1532	1538	
77	3840	84	1916	1922	
78	2560	36	1276	1282	
79	3072	36	1532	1538	
7A	3584	68	1788	1794	
7B	4096	68	2044	2050	
7C	4608	100	2300	2306	
7D	5120	100	2556	2562	
7E	6144	132	3068	3074	
7F	7680	132	3836	3842	



Table 197. I<sup>2</sup>C divider and hold values (continued)

	IBC7-0 (hex)	SCL divider (clocks)	SDA hold (clocks)	SCL hold (start)	SCL hold (stop)
MUL = 4	80	80	28	24	44
	81	88	28	28	48
	82	96	32	32	52
	83	104	32	36	56
	84	112	36	40	60
	85	120	36	44	64
	86	136	40	52	72
	87	160	40	64	84
	88	112	28	40	60
	89	128	28	48	68
	8A	144	36	56	76
	8B	160	36	64	84
	8C	176	44	72	92
	8D	192	44	80	100
	8E	224	52	96	116
	8F	272	52	120	140
	90	192	36	72	100
	91	224	36	88	116
	92	256	52	104	132
	93	288	52	120	148
	94	320	68	136	164
	95	352	68	152	180
	96	416	84	184	212
	97	512	84	232	260
	98	320	36	152	164
	99	384	36	184	196
	9A	448	68	216	228
	9B	512	68	248	260
	9C	576	100	280	292
	9D	640	100	312	324
	9E	768	132	376	388
	9F	960	132	472	484
A0	640	68	312	324	
A1	768	68	376	388	
A2	896	132	440	452	
A3	1024	132	504	516	
A4	1152	196	568	580	
A5	1280	196	632	644	
A6	1536	260	760	772	
A7	1920	260	952	964	
A8	1280	132	632	644	
A9	1536	132	760	772	
AA	1792	260	888	900	
AB	2048	260	1016	1028	
AC	2304	388	1144	1156	
AD	2560	388	1272	1284	
AE	3072	516	1528	1540	
AF	3840	516	1912	1924	
30	2560	260	1272	1284	
B1	3072	260	1528	1540	
B2	3584	516	1784	1796	
B3	4096	516	2040	2052	
B4	4608	772	2296	2308	
B5	5120	772	2552	2564	
B6	6144	1028	3064	3076	
B7	7680	1028	3832	3844	
B8	5120	516	2552	2564	
B9	6144	516	3064	3076	
BA	7168	1028	3576	3588	
BB	8192	1028	4088	4100	
BC	9216	1540	4600	4612	
BD	10240	1540	5112	5124	
BE	12288	2052	6136	6148	
BF	15360	2052	7672	7684	

### 22.3.4 I<sup>2</sup>C Bus Control Register (IBCR)

Offset 0x2

Access: Read/write any time

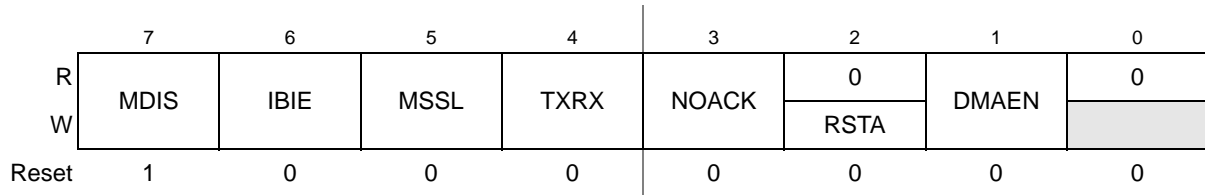


Figure 192. I<sup>2</sup>C Bus Control Register (IBCR)

Table 198. IBCR field descriptions

Field	Description
MDIS	<p>Module disable. This bit controls the software reset of the entire I<sup>2</sup>C Bus module.</p> <p>1 The module is reset and disabled. This is the power-on reset situation. When high, the interface is held in reset, but registers can still be accessed. Status register bits (IBSR) are not valid when module is disabled.</p> <p>0 The I<sup>2</sup>C Bus module is enabled. This bit must be cleared before any other IBCR bits have any effect</p> <p><b>Note:</b> If the I<sup>2</sup>C Bus module is enabled in the middle of a byte transfer, the interface behaves as follows: slave mode ignores the current transfer on the bus and starts operating whenever a subsequent start condition is detected. Master mode will not be aware that the bus is busy, hence if a start cycle is initiated then the current bus cycle may become corrupt. This would ultimately result in either the current bus master or the I<sup>2</sup>C Bus module losing arbitration, after which, bus operation would return to normal.</p>
IBIE	<p>I-Bus Interrupt Enable.</p> <p>1 Interrupts from the I<sup>2</sup>C Bus module are enabled. An I<sup>2</sup>C Bus interrupt occurs provided the IBIF bit in the status register is also set.</p> <p>0 Interrupts from the I<sup>2</sup>C Bus module are disabled. Note that this does not clear any currently pending interrupt condition</p>
MSSL	<p>Master/Slave mode select. Upon reset, this bit is cleared. When this bit is changed from 0 to 1, a START signal is generated on the bus and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. A STOP signal should be generated only if the IBIF flag is set. MSSL is cleared without generating a STOP signal when the master loses arbitration.</p> <p>1 Master Mode 0 Slave Mode</p>
TXRX	<p>Transmit/Receive mode select. This bit selects the direction of master and slave transfers. When addressed as a slave this bit should be set by software according to the SRW bit in the status register. In master mode this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always be high.</p> <p>1 Transmit 0 Receive</p>
NOACK	<p>Data Acknowledge disable. This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers. The I<sup>2</sup>C module will always acknowledge address matches, provided it is enabled, regardless of the value of NOACK. Note that values written to this bit are only used when the I<sup>2</sup>C Bus is a receiver, not a transmitter.</p> <p>1 No acknowledge signal response is sent (i.e., acknowledge bit = 1) 0 An acknowledge signal will be sent out to the bus at the 9th clock bit after receiving one byte of data</p>



**Table 198. IBCR field descriptions (continued)**

Field	Description
RSTA	Repeat Start. Writing a 1 to this bit will generate a repeated START condition on the bus, provided it is the current bus master. This bit will always be read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, will result in loss of arbitration. 1 Generate repeat start cycle 0 No effect
DMAEN	DMA Enable. When this bit is set, the DMA TX and RX lines will be asserted when the I <sup>2</sup> C module requires data to be read or written to the data register. No Transfer Done interrupts will be generated when this bit is set, however an interrupt will be generated if the loss of arbitration or addressed as slave conditions occur. The DMA mode is only valid when the I <sup>2</sup> C module is configured as a Master and the DMA transfer still requires CPU intervention at the start and the end of each frame of data. See the DMA Application Information section for more details. 1 Enable the DMA TX/RX request signals 0 Disable the DMA TX/RX request signals

### 22.3.5 I<sup>2</sup>C Bus Status Register (IBSR)

Offset 0x3

Access: Read-write

	7	6	5	4	3	2	1	0
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W				w1c			w1c	
Reset	1	0	0	0	0	0	0	0

**Figure 193. I<sup>2</sup>C Bus Status Register (IBSR)**

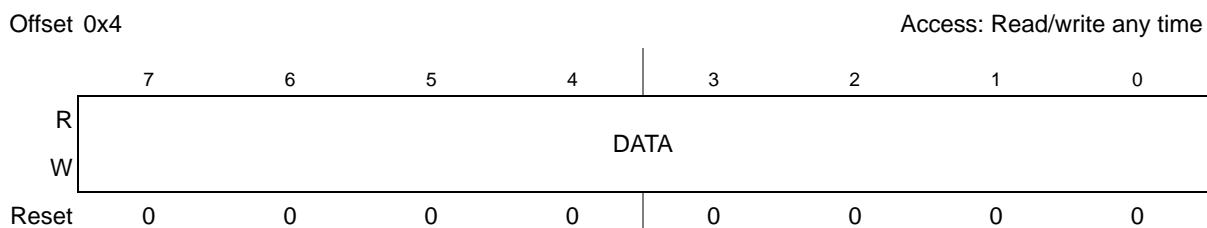
**Table 199. IBSR Field Descriptions**

Field	Description
TCF	Transfer complete. While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. Note that this bit is only valid during or immediately following a transfer to the I <sup>2</sup> C module or from the I <sup>2</sup> C module. 1 Transfer complete 0 Transfer in progress
IAAS	Addressed as a slave. When its own specific address (I-Bus Address Register) is matched with the calling address, this bit is set. The CPU is interrupted provided the IBIE is set. Then the CPU needs to check the SRW bit and set its Tx/Rx mode accordingly. Writing to the I-Bus Control Register clears this bit. 1 Addressed as a slave 0 Not addressed
IBB	Bus busy. This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and the bus enters idle state. 1 Bus is busy 0 Bus is Idle

**Table 199. IBSR Field Descriptions (continued)**

Field	Description
IBAL	Arbitration Lost. The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances: <ul style="list-style-type: none"> <li>– SDA is sampled low when the master drives a high during an address or data transmit cycle.</li> <li>– SDA is sampled low when the master drives a high during the acknowledge bit of a data receive cycle.</li> <li>– A start cycle is attempted when the bus is busy.</li> <li>– A repeated start cycle is requested in slave mode.</li> <li>– A stop condition is detected when the master did not request it.</li> </ul>
SRW	Slave Read/Write. When IAAS is set, this bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is only valid when the I-Bus is in slave mode, a complete address transfer has occurred with an address match and no other transfers have been initiated. By programming this bit, the CPU can select slave transmit/receive mode according to the command of the master. <ul style="list-style-type: none"> <li>1 Slave transmit, master reading from slave</li> <li>0 Slave receive, master writing to slave</li> </ul>
IBIF	I-Bus Interrupt Flag. The IBIF bit is set when one of the following conditions occurs: <ul style="list-style-type: none"> <li>– Arbitration lost (IBAL bit set)</li> <li>– Byte transfer complete (TCF bit set - Check w/ design if this is the case (only TCF))</li> <li>– Addressed as slave (IAAS bit set)</li> <li>– NoAck from Slave (MS &amp; Tx bits set)</li> <li>– I<sup>2</sup>C Bus going idle (IBB high-low transition and enabled by BIIE)</li> </ul> A processor interrupt request will be caused if the IBIE bit is set.
RXAK	Received Acknowledge. This is the value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock. This bit is valid only after transfer is complete. <ul style="list-style-type: none"> <li>1 No acknowledge received</li> <li>0 Acknowledge received</li> </ul>

### 22.3.6 I<sup>2</sup>C Bus Data I/O Register (IBDR)



**Figure 194. I<sup>2</sup>C Bus Data I/O Register (IBDR)**

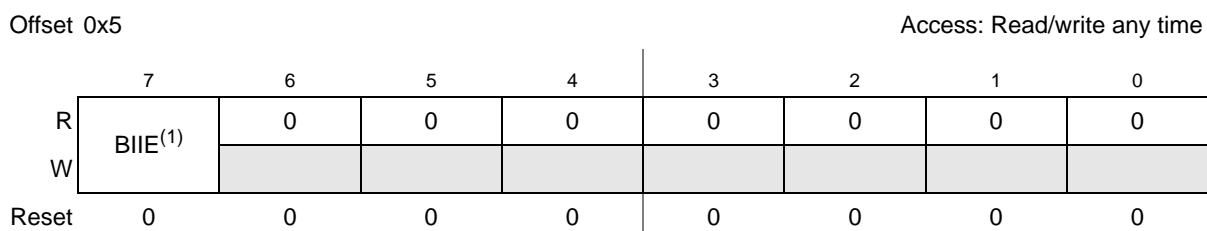
In master transmit mode, when data is written to IBDR, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same functions are available after an address match has occurred. Note that the IBCR[TRX] field must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the I<sup>2</sup>C is

configured for master transmit but a master receive is desired, then reading the IBDR will not initiate the receive.

Reading the IBDR will return the last byte received while the I<sup>2</sup>C is configured in either master receive or slave receive modes. The IBDR does not reflect every byte that is transmitted on the I<sup>2</sup>C bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master transmit mode, the first byte of data written to IBDR following assertion of MS/SL is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required R/W bit (in position D0).

### 22.3.7 I<sup>2</sup>C Bus Interrupt Config Register (IBIC)



**Figure 195. I<sup>2</sup>C Bus Interrupt Config Register (IBIC)**

1. This bit cannot be set in reset state, when I2C is in slave mode. It can be set to 1 only when I2C is in Master mode. This information is missing from the spec.

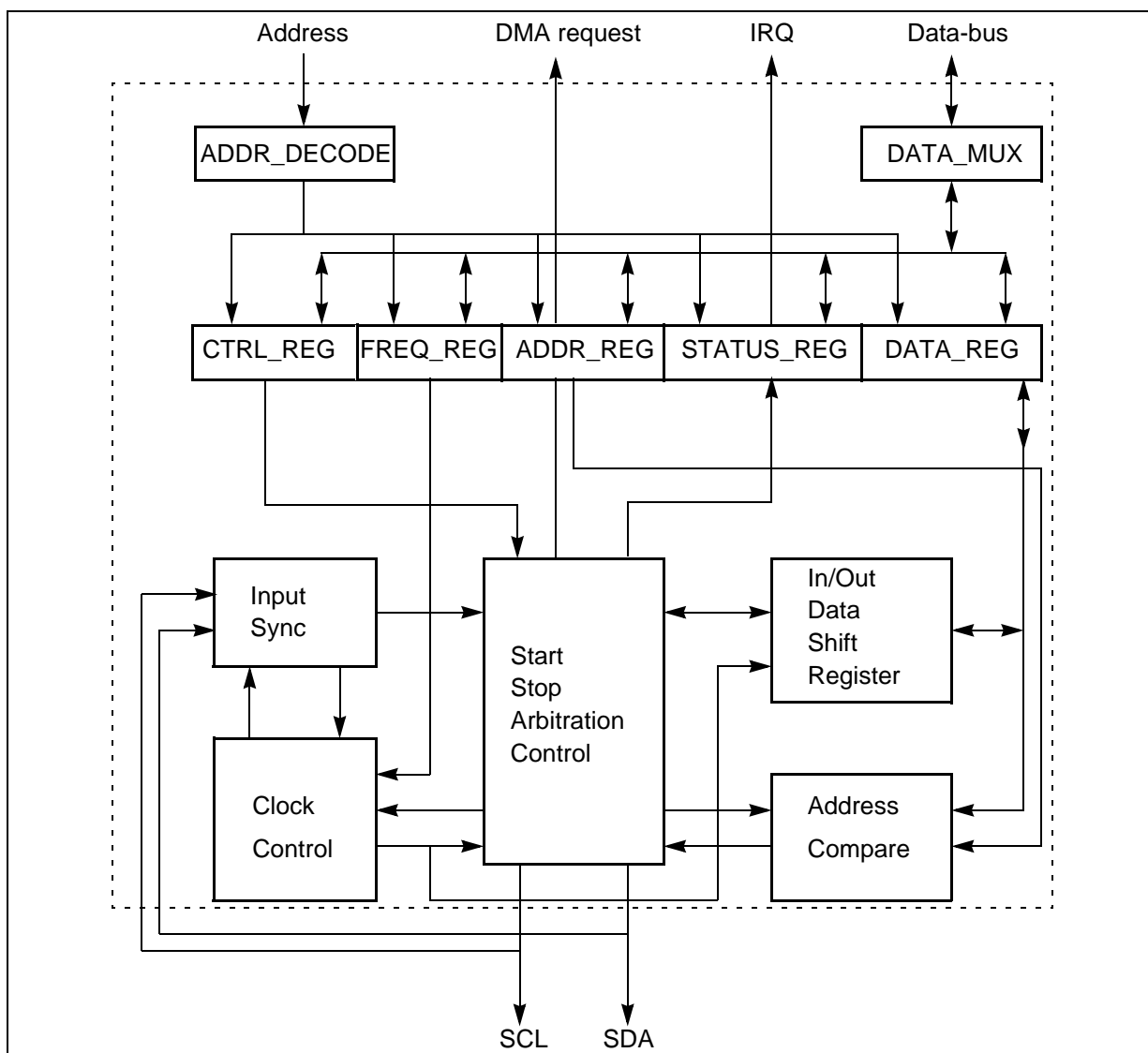
**Table 200. IBIC field descriptions**

Field	Description
BIIE	Bus Idle Interrupt Enable bit. This config bit can be used to enable the generation of an interrupt once the I <sup>2</sup> C bus becomes idle. Once this bit is set, an IBB high-low transition will set the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I <sup>2</sup> C bus. 1 Bus Idle Interrupts enabled 0 Bus Idle Interrupts disabled

### 22.4 DMA Interface

A simple DMA interface is implemented so that the I<sup>2</sup>C can request data transfers with minimal support from the CPU. DMA mode is enabled by setting bit 1 in the Control Register.

The DMA interface is only valid when the I<sup>2</sup>C module is configured for Master Mode.



**Figure 196. I<sup>2</sup>C module DMA interface block diagram**

At least 3 bytes of data per frame must be transferred from/to the slave when using DMA mode, although in practice it will only be worthwhile using the DMA mode when there is a large number of data bytes to transfer per frame.

Two internal signals, TX request and RX request, are used to signal to a DMA controller when the I<sup>2</sup>C module requires data to be written or read from the data register.

## 22.5 Functional description

### 22.5.1 I-Bus protocol

The I<sup>2</sup>C Bus system uses a Serial Data line (SDA) and a Serial Clock Line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. A

logical AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer and STOP signal. They are described briefly in the following sections and illustrated in [Figure 197](#).

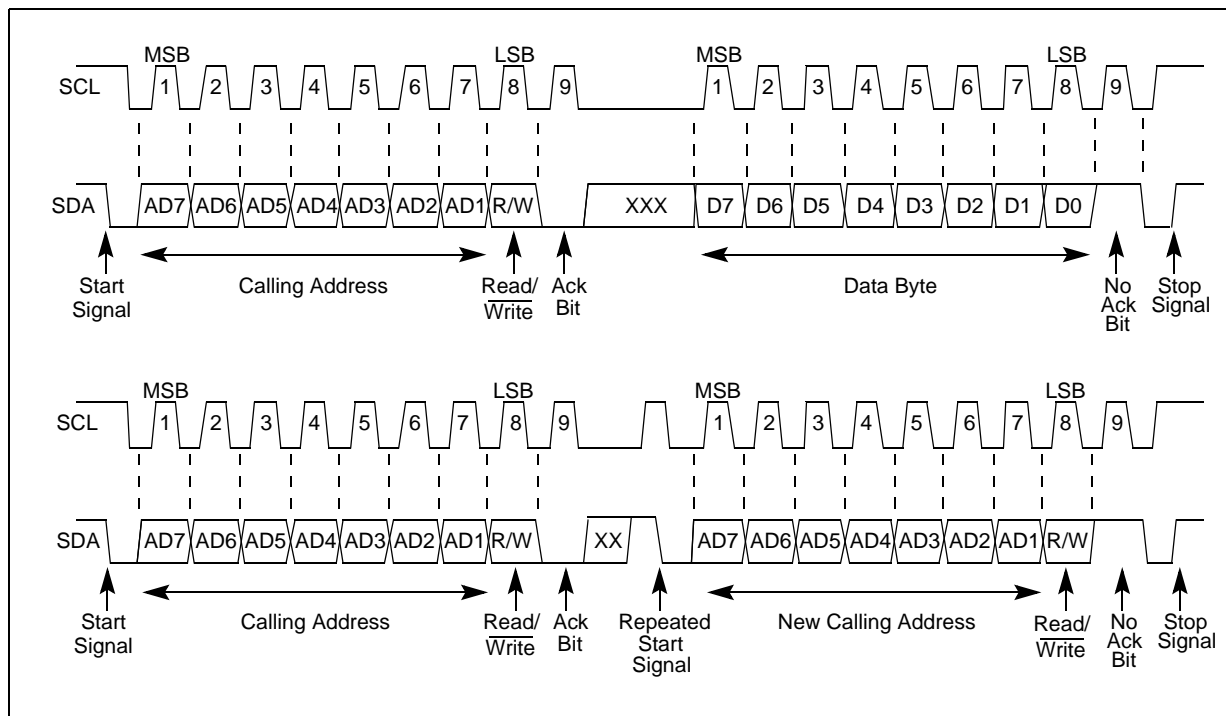


Figure 197. I<sup>2</sup>C bus transmission signals

### 22.5.1.1 START signal

When the bus is free, i.e. no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in [Figure 197](#), a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

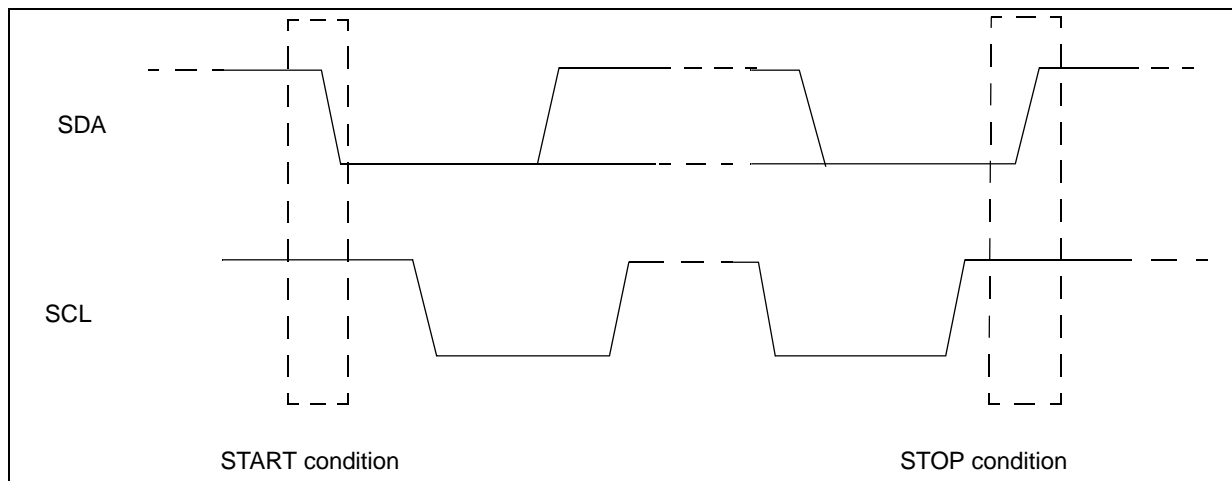


Figure 198. Start and stop conditions

### 22.5.1.2 Slave address transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer - the slave transmits data to the master

0 = Write transfer - the master transmits data to the slave

Only the slave with a calling address that matches the one transmitted by the master will respond by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see [Figure 197](#)).

No two slaves in the system may have the same address. If the I<sup>2</sup>C Bus is master, it must not transmit an address that is equal to its own slave address. The I<sup>2</sup>C Bus cannot be master and slave at the same time. However, if arbitration is lost during an address cycle the I<sup>2</sup>C Bus will revert to slave mode and operate correctly, even if it is being addressed by another master.

### 22.5.1.3 Data transfer

Once successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in [Figure 197](#). There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA low at the ninth clock. Therefore, one complete data byte transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means 'end of data' to the slave, so the slave releases the SDA line for the master to generate a STOP or START signal.

#### 22.5.1.4 STOP signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical "1" (see [Figure 197](#)).

The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

#### 22.5.1.5 Repeated START signal

As shown in [Figure 197](#), a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

#### 22.5.1.6 Arbitration procedure

The Inter-IC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure. A bus master loses arbitration if it transmits logic "1" while another master transmits logic "0". The losing masters immediately switch over to slave receive mode and stop driving the SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

#### 22.5.1.7 Clock synchronization

Since wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see [Figure 199](#)). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

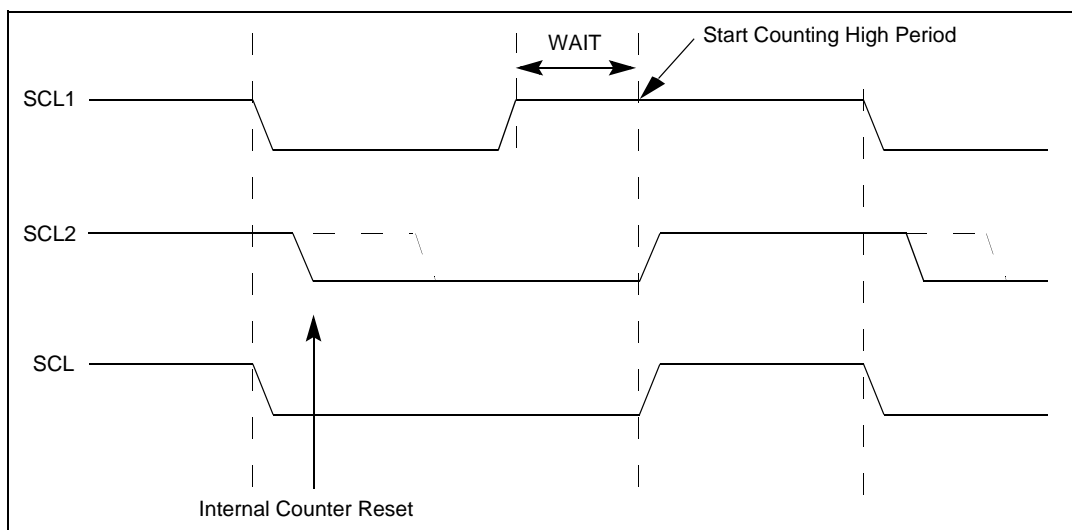


Figure 199. I<sup>2</sup>C bus clock synchronization

**22.5.1.8 Handshaking**

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait state until the slave releases the SCL line.

**22.5.1.9 Clock stretching**

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

**22.5.2 Interrupts**

**22.5.2.1 General**

The I<sup>2</sup>C uses only one interrupt vector.

Table 201. Interrupt summary

Interrupt	Offset	Vector	Priority	Source	Description
I <sup>2</sup> C Interrupt	—	—	—	IBAL, TCF, IAAS, IBB bits in IBSR register	When any of IBAL, TCF or IAAS bits is set an interrupt may be caused based on Arbitration lost, Transfer Complete or Address Detect conditions. If enabled by BIIE, the deassertion of IBB can also cause an interrupt, indicating that the bus is idle.

**22.5.2.2 Interrupt description**

There are five types of internal interrupts in the I<sup>2</sup>C. The interrupt service routine can determine the interrupt type by reading the Status Register.



I<sup>2</sup>C Interrupt can be generated on

- Arbitration Lost condition (IBAL bit set)
- Byte Transfer condition (TCF bit set and DMAEN bit not set)
- Address Detect condition (IAAS bit set)
- No Acknowledge from slave received when expected
- Bus Going Idle (IBB bit not set)

The I<sup>2</sup>C interrupt is enabled by the IBIE bit in the I<sup>2</sup>C Control Register. It must be cleared by writing '1' to the IBIF bit in the interrupt service routine. The Bus Going Idle interrupt needs to be additionally enabled by the BIIE bit in the IBIC register.

## 22.6 Initialization/application information

### 22.6.1 I<sup>2</sup>C programming examples

#### 22.6.1.1 Initialization sequence

Reset will put the I<sup>2</sup>C Bus Control Register to its default state. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the Frequency Divider Register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the I<sup>2</sup>C Bus Address Register (IBAD) to define its slave address.
3. Clear the IBCR[MDIS] field to enable the I<sup>2</sup>C interface system.
4. Modify the bits of the I<sup>2</sup>C Bus Control Register (IBCR) to select Master/Slave mode, Transmit/Receive mode and interrupt enable or not. Optionally also modify the bits of the I<sup>2</sup>C Bus Interrupt Config Register (IBIC) to further refine the interrupt behavior.

#### 22.6.1.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the 'master transmitter' mode. If the device is connected to a multi-master bus system, the state of the I<sup>2</sup>C Bus Busy bit (IBB) must be tested to check whether the serial bus is free.

If the bus is free (IBB=0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB, which is set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I<sup>2</sup>C is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of the sequence of events which generates the START signal and transmits the first byte of data (slave address) is shown below:

```
while (bit 5, IBSR ==1)// wait in loop for IBB flag to clear
bit4 and bit 5, IBCR = 1// set transmit and master mode, i.e. generate
start condition
IBDR = calling_address// send the calling address to the data register
while (bit 5, IBSR ==0)// wait in loop for IBB flag to be set
```

### 22.6.1.3 Post-transfer software response

Transmission or reception of a byte will set the data transferring bit (TCF) to 1, which indicates one byte communication is finished. The I<sup>2</sup>C Bus interrupt bit (IBIF) is set also; an interrupt will be generated if the interrupt function is enabled during initialization by setting the IBIE bit. The IBIF (interrupt flag) can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit will be cleared to indicate data transfer in progress whenever data register is written to in transmit mode, or during reading out from data register in receive mode. The TCF bit should not be used as a data transfer complete flag as the flag timing is dependent on a number of factors including the I<sup>2</sup>C bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. It is recommended that transfer complete situations are detected using the IBIF flag

Software may service the I<sup>2</sup>C I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Note that polling should monitor the IBIF bit rather than the TCF bit since their operation is different when arbitration is lost.

Note that when a "Transfer Complete" interrupt occurs at the end of the address cycle, the master will always be in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/W bit sent with slave calling address, then the Tx/Rx bit at Master side should be toggled at this stage. If Master does not receive an ACK from Slave, then transmission must be re-initiated or terminated.

In slave mode, IAAS bit will get set in IBSR if Slave address (IBAD) matches the Master calling address. This is an indication that Master-Slave data communication can now start. During address cycles (IAAS=1), the SRW bit in the status register is read to determine the direction of the subsequent transfer and the Tx/Rx bit is programmed accordingly. For slave mode data cycles (IAAS=0), the SRW bit is not valid. The Tx/Rx bit in the control register should be read to determine the direction of the current transfer.

### 22.6.1.4 Transmit/receive sequence

Follow this sequence in case of Master Transmit(Address/Data):

1. Clear IBSR[IBIF].
2. Write data in Data Register (IBDR).
3. IBSR[TCF] bit will get cleared when transfer is in progress.
4. IBSR[TCF] bit will get set when transfer is complete.
5. Wait for IBSR[IBIF] to get set, then read IBSR register to determine its source:
  - TCF = 1 i.e. transfer is complete.
  - No Acknowledge condition (RXAK = 1) is found.
  - IBB = 0 i.e. Bus has transitioned from Busy to Idle state.
  - If IBB = 1, ignore check of Arbitration Loss (IBAL = 1).
  - Ignore Address Detect (IAAS = 1) for Master mode (valid only for Slave mode).
6. Check RXAK in IBSR for an acknowledge from slave.

Follow this sequence in case of Slave Receive(Address/Data):

1. Clear IBSR[IBIF].
2. IBSR[TCF] will get cleared when transfer is in progress for address transfer.
3. IBSR[TCF] will get set when transfer is complete.
4. Wait for IBSR[IBIF] to get set. Then read IBSR register to determine its source:
  - Address Detect has occurred (IAAS = 1) - determination of Slave mode.
5. Clear IBIF.
6. Wait until IBSR[TCF] bit gets cleared (that is, "Transfer under Progress" condition is reached for data transfer).
7. Wait until IBSR[TCF] bit gets cleared (proof that Transfer Completes from "Transfer under Progress" state).
8. Wait until IBSR[IBIF] bit gets set. To find its source, check if:
  - TCF = 1 i.e. reception is complete
  - IBSR[IBB] = 0, that is, bus has transitioned from Busy to Idle state
  - Ignore Arbitration Loss (IBAL = 1) for IBB = 1
  - Ignore No Acknowledge condition (RXAK = 1) for receiver
9. Read the Data Register (IBDR) to determine data received from Master.

Sequence followed in case of Slave Transmit (Steps 1–4 of Slave Receive for Address Detect, followed by 1–6 of Master Transmit for Data Transmit).

Sequence followed in case of Master Receive (Steps 1–6 of Master Transmit for Address dispatch, followed by 5–8 of Slave Receive for Data Receive).

### 22.6.1.5 Generation of STOP

A data transfer ends with a STOP signal generated by the 'master' device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following is an example showing how a stop condition is generated by a master transmitter.

```

if (tx_count == 0) or// check to see if all data bytes have been
transmitted
    (bit 0, IBSR == 1) {// or if no ACK generated
        clear bit 5, IBCR// generate stop condition
    }
else {
    IBDR = data_to_transmit// write byte of data to DATA register
    tx_count --// decrement counter
} // return from interrupt

```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data which can be done by setting the NOACK bit in IBCR before reading the 2nd last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following is an example showing how a STOP signal is generated by a master receiver.

```

rx_count --// decrease the rx counter
if (rx_count ==1)// 2nd last byte to be read ?
    bit 3, IBCR = 1// disable ACK
if (rx_count == 0)// last byte to be read ?
    bit 5, IBCR = 0// generate stop signal
else
    data_received = IBDR// read RX data and store

```

### 22.6.1.6 Generation of repeated START

At the end of data transfer, if the master still wants to communicate on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```
bit 2, IBCR = 1// generate another start ( restart)
IBDR == calling_address// transmit the calling address
```

### 22.6.1.7 Slave mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) should be tested to check if a calling of its own address has just been received. If IAAS is set, software should set the transmit/receive mode select bit (Tx/Rx bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears IAAS automatically. Note that the only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred. Interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer may now be initiated by writing information to IBDR for slave transmits or dummy reading from IBDR in slave receive mode. The slave will drive SCL low in-between byte transfers SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an 'end of data' signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so that the master can generate a STOP signal.

### 22.6.1.8 Arbitration lost

If several masters try to engage the bus simultaneously, only one master wins and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL is still generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL=1 and MS/SL=0. If one master attempts to start transmission, while the bus is being engaged by another master, the hardware will inhibit the transmission, switch the MS/SL bit from 1 to 0 without generating a STOP condition, generate an interrupt to CPU and set the IBAL to indicate that the attempt to engage the bus is failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

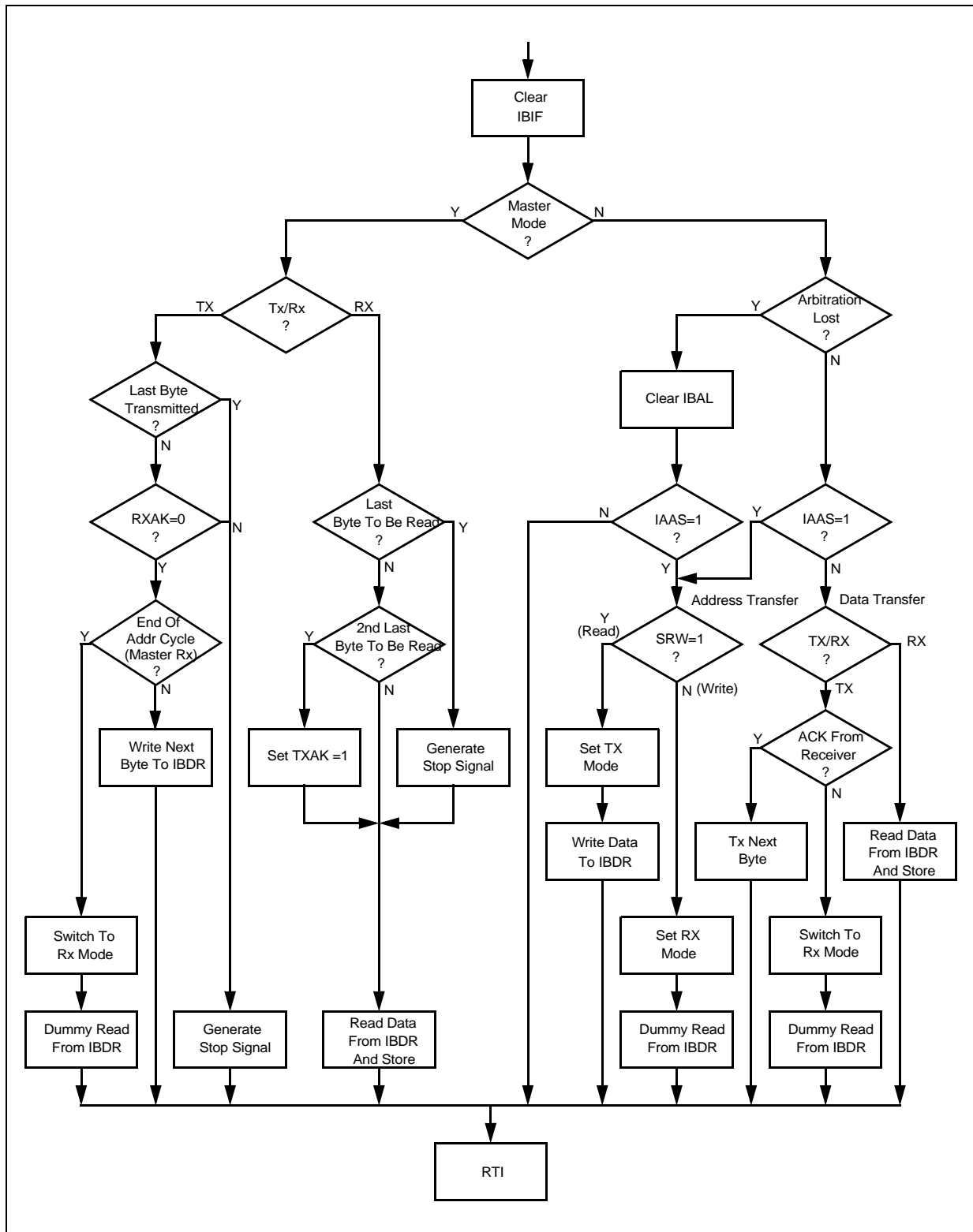


Figure 200. Flow-Chart of Typical I<sup>2</sup>C Interrupt Routine

## 23 LIN Controller (LINFlex)

### 23.1 Introduction

The LINFlex (Local Interconnect Network Flexible) controller interfaces the LIN network and supports the LIN protocol versions 1.3; 2.0 and 2.1; and J2602 in both Master and Slave modes. LINFlex includes a LIN mode that provides additional features (compared to standard UART) to ease LIN implementation, improve system robustness, minimize CPU load and allow slave node resynchronization.

### 23.2 Main features

#### 23.2.1 LIN mode features

- Supports LIN protocol versions 1.3, 2.0, 2.1 and J2602
- Master mode with autonomous message handling
- Classic and enhanced checksum calculation and check
- Single 8-byte buffer for transmission/reception
- Extended frame mode for In-Application Programming (IAP) purposes
- Wake-up event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with 16 MHz fast internal RC oscillator as clock source
- 16 identifier filters for autonomous message handling in Slave mode
- Peripheral DMA request sources possible from LINFlex

#### 23.2.2 UART mode features

- Full duplex communication
- 8- or 9-bit with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

### 23.2.3 Features common to LIN and UART

- Fractional baud rate generator
- 3 operating modes for power saving and configuration registers lock:
  - Initialization
  - Normal
  - Sleep
- 2 test modes:
  - Loop Back
  - Self Test
- Maskable interrupts

## 23.3 General description

The increasing number of communication peripherals embedded on microcontrollers, for example CAN, LIN and SPI, requires more and more CPU resources for communication management. Even a 32-bit microcontroller is overloaded if its peripherals do not provide high-level features to autonomously handle the communication.

Even though the LIN protocol with a maximum baud rate of 20 Kbit/s is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as usually the case.

To minimize the CPU load in Master mode, LINFlex handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlex does not request any software intervention until the next header transmission request in transmission mode or until the checksum reception in reception mode.

To minimize the CPU load in Slave mode, LINFlex requires software intervention only to:

- Trigger transmission or reception or data discard depending on the identifier
- Write data into the buffer (transmission mode) or read data from the buffer (reception mode) after checksum reception

If filter mode is activated for Slave mode, LINFlex requires software intervention only to write data into the buffer (transmission mode) or read data from the buffer (reception mode)

The software uses the control, status and configuration registers to:

- Configure LIN parameters (for example, baud rate or mode)
- Request transmissions
- Handle receptions
- Manage interrupts
- Configure LIN error and timeout detection
- Process diagnostic information

The message buffer stores transmitted or received LIN frames.

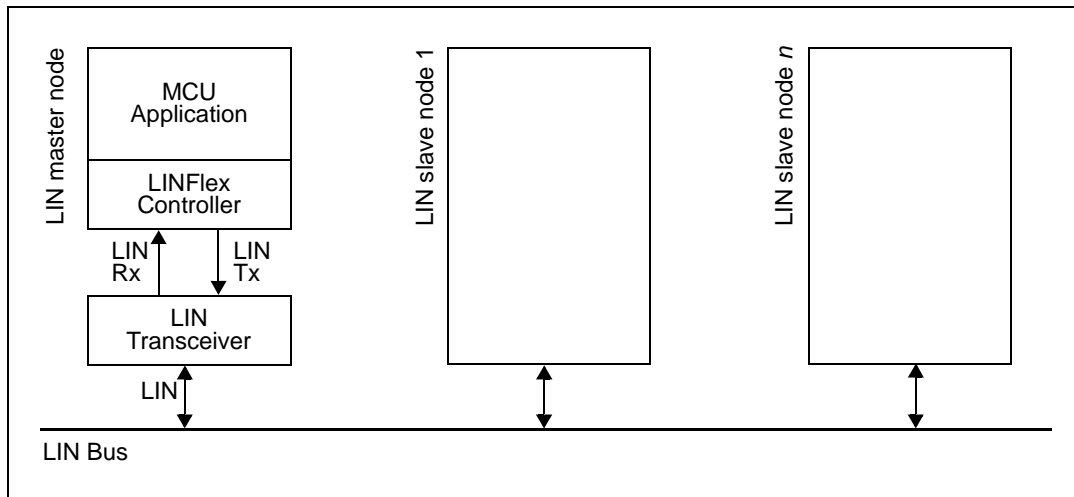


Figure 201. LIN topology network

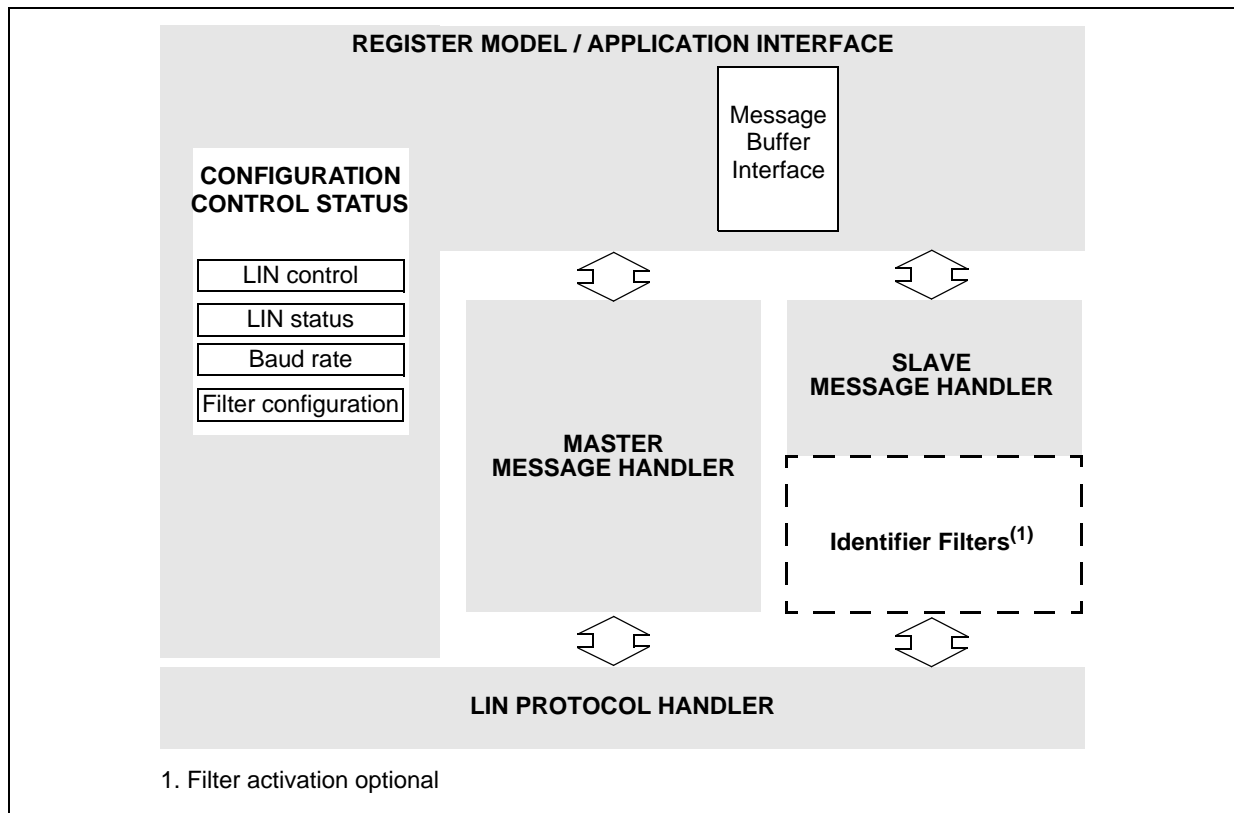


Figure 202. LINFlex block diagram

### 23.4 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in the Mantissa (LINIBRR) and Fraction (LINFBR) registers.



**Equation 7**

$$\text{Tx/ Rx baud} = \frac{f_{\text{periph\_set\_1\_clk}}}{(16 \times \text{LFDIV})}$$

LFDIV is an unsigned fixed point number. The 12-bit mantissa is coded in the LINIBRR and the fraction is coded in the LINFBR.

The following examples show how to derive LFDIV from LINIBRR and LINFBR register values:

**Example 4** Deriving LFDIV from LINIBRR and LINFBR register values

If LINIBRR = 27d and LINFBR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

**Example 5** Programming LFDIV from LINIBRR and LINFBR register values

To program LFDIV = 25.62d,

LINFBR = 16 × 0.62 = 9.92, nearest real number 10d = 0xA

LINIBRR = mantissa (25.620d) = 25d = 0x19

*Note:* The baud counters are updated with the new value of the baud registers after a write to LINIBRR. Hence the baud register value must not be changed during a transaction. The LINFBR (containing the Fraction bits) must be programmed before the LINIBRR.

*Note:* LFDIV must be greater than or equal to 1.5d, i.e. LINIBRR = 1 and LINFBR = 8. Therefore, the maximum possible baudrate is  $f_{\text{periph\_set\_1\_clk}} / 24$ .

**Table 202. Error calculation for programmed baud rates**

Baud rate	$f_{\text{periph\_set\_1\_clk}} = 64 \text{ MHz}$				$f_{\text{periph\_set\_1\_clk}} = 16 \text{ MHz}$			
	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate
		LINIBRR	LINFBR			LINIBRR	LINFBR	
2400	2399.97	1666	11	-0.001	2399.88	416	11	-0.005
9600	9599.52	416	11	-0.005	9598.08	104	3	-0.02
10417	10416.7	384	0	-0.003	10416.7	96	0	-0.003
19200	19201.9	208	5	0.01	19207.7	52	1	0.04
57600	57605.8	69	7	0.01	57554	17	6	-0.08
115200	115108	34	12	-0.08	115108	8	11	-0.08
230400	230216	17	6	-0.08	231884	4	5	0.644
460800	460432	8	11	-0.08	457143	2	3	-0.794
921600	927536	4	5	0.644	941176	1	1	2.124

## 23.5 Operating modes

LINFlex has three main operating modes: Initialization, Normal and Sleep. After a hardware reset, LINFlex is in Sleep mode to reduce power consumption. The software instructs LINFlex to enter Initialization mode or Sleep mode by setting the INIT bit or SLEEP bit in the LINCR1.

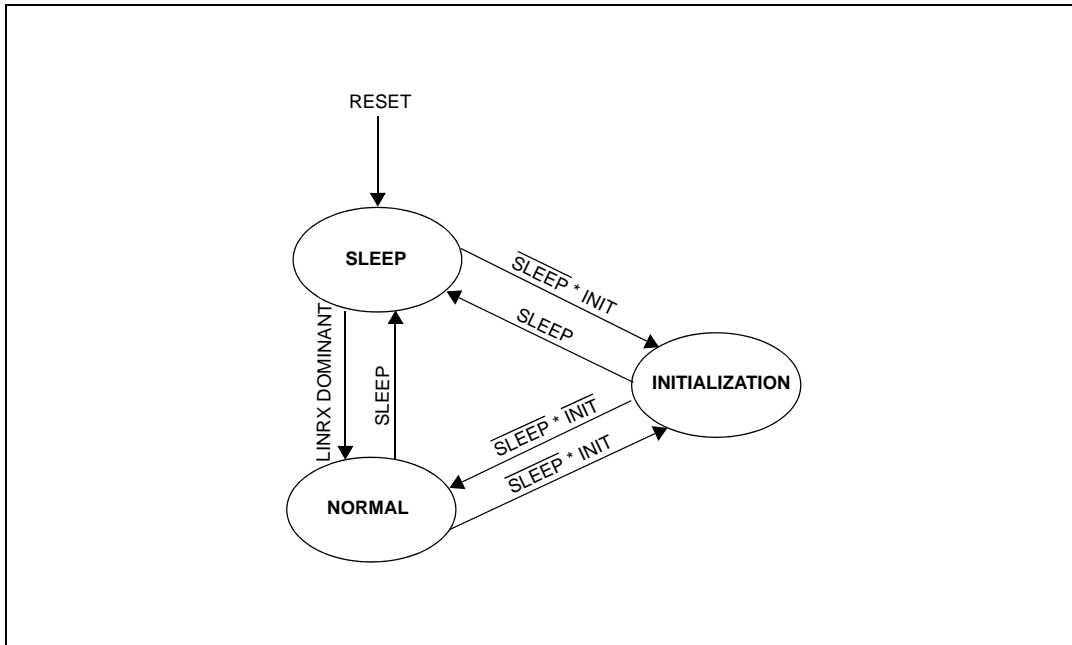


Figure 203. LINFlex operating modes

### 23.5.1 Initialization mode

The software can be initialized while the hardware is in Initialization mode. To enter this mode the software sets the INIT bit in the LINCR1.

To exit Initialization mode, the software clears the INIT bit.

While in Initialization mode, all message transfers to and from the LIN bus are stopped and the status of the LIN bus output LINTX is recessive (high).

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlex controller, the software selects the mode (LIN Master, LIN Slave or UART), sets up the baud rate register and, if LIN Slave mode with filter activation is selected, initializes the identifier list.

### 23.5.2 Normal mode

Once initialization is complete, software clears the INIT bit in the LINCR1 to put the hardware into Normal mode.

### 23.5.3 Low power mode (Sleep)

To reduce power consumption, LINFlex has a low power mode called Sleep mode. To enter Sleep mode, software sets the SLEEP bit in the LINCR1. In this mode, the LINFlex clock is stopped. Consequently, the LINFlex will not update the status bits but software can still access the LINFlex registers.

LINFlex can be awakened (exit Sleep mode) either by software clearing the SLEEP bit or on detection of LIN bus activity if automatic wake-up mode is enabled (AWUM bit is set).

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing the SLEEP bit if the AWUM bit in the LINCR1 is set. To exit from Sleep mode if the AWUM bit is cleared, software clears the SLEEP bit when a wake-up event occurs.

## 23.6 Test modes

Two test modes are available to the user: Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCR1. These bits must be configured while LINFlex is in Initialization mode. Once one of the two test modes has been selected, LINFlex must be started in Normal mode.

### 23.6.1 Loop Back mode

LINFlex can be put in Loop Back mode by setting the LBKM bit in the LINCR. In Loop Back mode, the LINFlex treats its own transmitted messages as received messages.

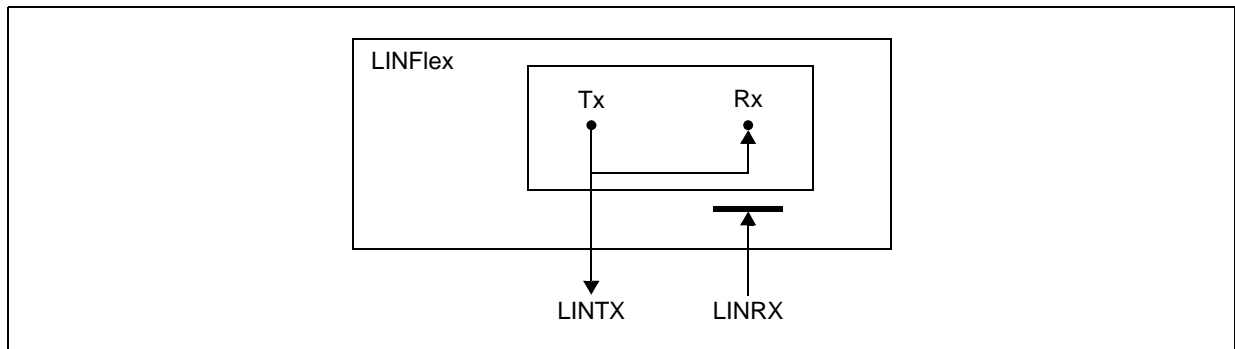


Figure 204. LINFlex in loop back mode

This mode is provided for self test functions. To be independent of external events, the LIN core ignores the LINRX signal. In this mode, the LINFlex performs an internal feedback from its Tx output to its Rx input. The actual value of the **LINRX** input pin is disregarded by the LINFlex. The transmitted messages can be monitored on the **LINTX** pin.

### 23.6.2 Self Test mode

LINFlex can be put in Self Test mode by setting the LBKM and SFTM bits in the LINCR. This mode can be used for a “Hot Self Test”, meaning the LINFlex can be tested as in Loop Back mode but without affecting a running LIN system connected to the LINTX and LINRX pins. In this mode, the LINRX pin is disconnected from the LINFlex and the LINTX pin is held recessive.

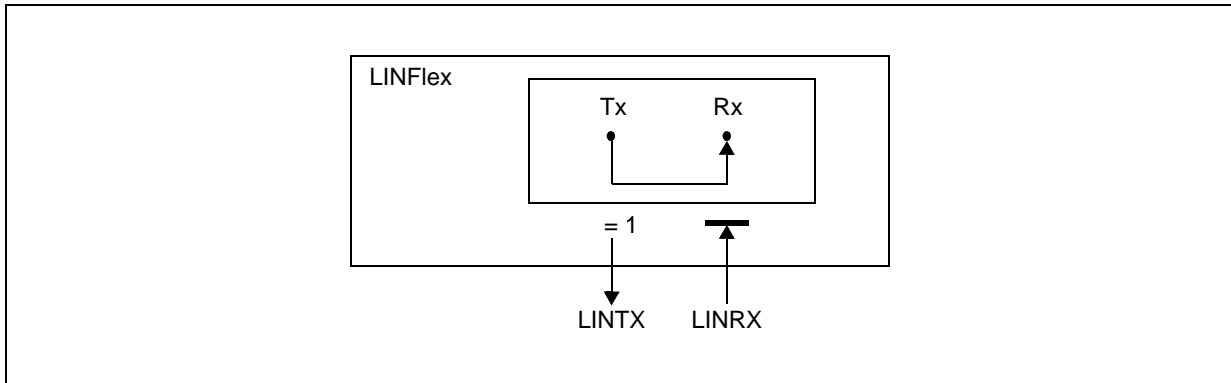


Figure 205. LINFlex in self test mode

## 23.7 Memory map and registers description

### 23.7.1 Memory map

See the “Memory map” chapter of this reference manual for the base addresses for the LINFlex modules.

Table 203 shows the LINFlex memory map.

Table 203. LINFlex memory map

Address offset	Register	Location
0x0000	LIN control register 1 (LINC1R1)	<a href="#">on page 469</a>
0x0004	LIN interrupt enable register (LINIER)	<a href="#">on page 472</a>
0x0008	LIN status register (LINSR)	<a href="#">on page 474</a>
0x000C	LIN error status register (LINESR)	<a href="#">on page 477</a>
0x0010	UART mode control register (UARTCR)	<a href="#">on page 478</a>
0x0014	UART mode status register (UARTSR)	<a href="#">on page 480</a>
0x0018	LIN timeout control status register (LINTCSR)	<a href="#">on page 482</a>
0x001C	LIN output compare register (LINOOCR)	<a href="#">on page 483</a>
0x0020	LIN timeout control register (LINTOCR)	<a href="#">on page 483</a>
0x0024	LIN fractional baud rate register (LINFBR)	<a href="#">on page 484</a>
0x0028	LIN integer baud rate register (LINIBRR)	<a href="#">on page 485</a>
0x002C	LIN checksum field register (LINCFCR)	<a href="#">on page 486</a>
0x0030	LIN control register 2 (LINC1R2)	<a href="#">on page 486</a>
0x0034	Buffer identifier register (BIDR)	<a href="#">on page 488</a>
0x0038	Buffer data register LSB (BDRL) <sup>(1)</sup>	<a href="#">on page 489</a>
0x003C	Buffer data register MSB (BDRM) <sup>(2)</sup>	<a href="#">on page 490</a>
0x0040	Identifier filter enable register (IFER)	<a href="#">on page 490</a>

**Table 203. LINFlex memory map (continued)**

Address offset	Register	Location
0x0044	Identifier filter match index (IFMI)	<a href="#">on page 492</a>
0x0048	Identifier filter mode register (IFMR)	<a href="#">on page 492</a>
0x004C	Identifier filter control register 0 (IFCR0)	<a href="#">on page 494</a>
0x0050	Identifier filter control register 1 (IFCR1)	<a href="#">on page 495</a>
0x0054	Identifier filter control register 2 (IFCR2)	<a href="#">on page 495</a>
0x0058	Identifier filter control register 3 (IFCR3)	<a href="#">on page 495</a>
0x005C	Identifier filter control register 4 (IFCR4)	<a href="#">on page 495</a>
0x0060	Identifier filter control register 5 (IFCR5)	<a href="#">on page 495</a>
0x0064	Identifier filter control register 6 (IFCR6)	<a href="#">on page 495</a>
0x0068	Identifier filter control register 7 (IFCR7)	<a href="#">on page 495</a>
0x006C	Identifier filter control register 8 (IFCR8)	<a href="#">on page 495</a>
0x0070	Identifier filter control register 9 (IFCR9)	<a href="#">on page 495</a>
0x0074	Identifier filter control register 10 (IFCR10)	<a href="#">on page 495</a>
0x0078	Identifier filter control register 11 (IFCR11)	<a href="#">on page 495</a>
0x007C	Identifier filter control register 12 (IFCR12)	<a href="#">on page 495</a>
0x0080	Identifier filter control register 13 (IFCR13)	<a href="#">on page 495</a>
0x0084	Identifier filter control register 14 (IFCR14)	<a href="#">on page 495</a>
0x0088	Identifier filter control register 15 (IFCR15)	<a href="#">on page 495</a>
0x008C–0x000F	Reserved	

1. LSB: Least significant byte.
2. MSB: Most significant byte.

**23.7.1.1 LIN control register 1 (LINCRI)**

Offset: 0x0000

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CCD	CFD	LASE	AWUM	MBL				BF	SFT M	LBK M	MME	SBD T	RBL M	SLEEP	INI T
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

**Figure 206. LIN control register 1 (LINCRI)**



**Table 204. LINCR1 field descriptions**

Field	Description
CCD	<p>Checksum calculation disable                      This bit disables the checksum calculation (see <a href="#">Table 205</a>).</p> <p>0 Checksum calculation is done by hardware. When this bit is 0, the LINCFR is read-only.                      1 Checksum calculation is disabled. When this bit is set the LINCFR is read/write. User can program this register to send a software-calculated CRC (provided CFD is 0).</p> <p><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>
CFD	<p>Checksum field disable                      This bit disables the checksum field transmission (see <a href="#">Table 205</a>).</p> <p>0 Checksum field is sent after the required number of data bytes is sent.                      1 No checksum field is sent.</p> <p><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>
LASE	<p>LIN Slave Automatic Resynchronization Enable                      0 Automatic resynchronization disable.                      1 Automatic resynchronization enable.</p> <p><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>
AWUM	<p>Automatic Wake-Up Mode                      This bit controls the behavior of the LINFlex hardware during Sleep mode.</p> <p>0 The Sleep mode is exited on software request by clearing the SLEEP bit of the LINCR.                      1 The Sleep mode is exited automatically by hardware on LINRX dominant state detection. The SLEEP bit of the LINCR is cleared by hardware whenever WUF bit in the LINSR is set.</p> <p><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>
MBL	<p>LIN Master Break Length                      This field indicates the Break length in Master mode (see <a href="#">Table 206</a>).</p> <p><b>Note:</b> This field can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>
BF	<p>Bypass filter                      0 No interrupt if identifier does not match any filter.                      1 An RX interrupt is generated on identifier not matching any filter.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>– If no filter is activated, this bit is reserved and always reads 1.</li> <li>– This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</li> </ul>
SFTM	<p>Self Test Mode                      This bit controls the Self Test mode. For more details, see <a href="#">Section 23.6.2: Self Test mode</a>.</p> <p>0 Self Test mode disable.                      1 Self Test mode enable.</p> <p><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>
LBKM	<p>Loop Back Mode                      This bit controls the Loop Back mode. For more details see <a href="#">Section 23.6.1: Loop Back mode</a>.</p> <p>0 Loop Back mode disable.                      1 Loop Back mode enable.</p> <p><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>
MME	<p>Master Mode Enable                      0 Slave mode enable.                      1 Master mode enable.</p> <p><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</p>

**Table 204. LINC1 field descriptions (continued)**

Field	Description
SBDT	Slave Mode Break Detection Threshold 0 11-bit break. 1 10-bit break. <b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
RBLM	Receive Buffer Locked Mode 0 Receive Buffer not locked on overrun. Once the Slave Receive Buffer is full the next incoming message overwrites the previous one. 1 Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded. <b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SLEEP	Sleep Mode Request This bit is set by software to request LINFlex to enter Sleep mode. This bit is cleared by software to exit Sleep mode or by hardware if the AWUM bit in LINC1 and the WUF bit in LINSR are set (see <a href="#">Table 207</a> ).
INIT	Initialization Request The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlex enters Normal mode when clearing the INIT bit (see <a href="#">Table 207</a> ).

**Table 205. Checksum bits configuration**

CFD	CCD	LINC1R	Checksum sent
1	1	Read/Write	None
1	0	Read-only	None
0	1	Read/Write	Programmed in LINC1R by bits CF[0:7]
0	0	Read-only	Hardware calculated

**Table 206. LIN master break length selection**

MBL	Length
0000	10-bit
0001	11-bit
0010	12-bit
0011	13-bit
0100	14-bit
0101	15-bit
0110	16-bit
0111	17-bit
1000	18-bit
1001	19-bit
1010	20-bit

**Table 206. LIN master break length selection (continued)**

MBL	Length
1011	21-bit
1100	22-bit
1101	23-bit
1110	36-bit
1111	50-bit

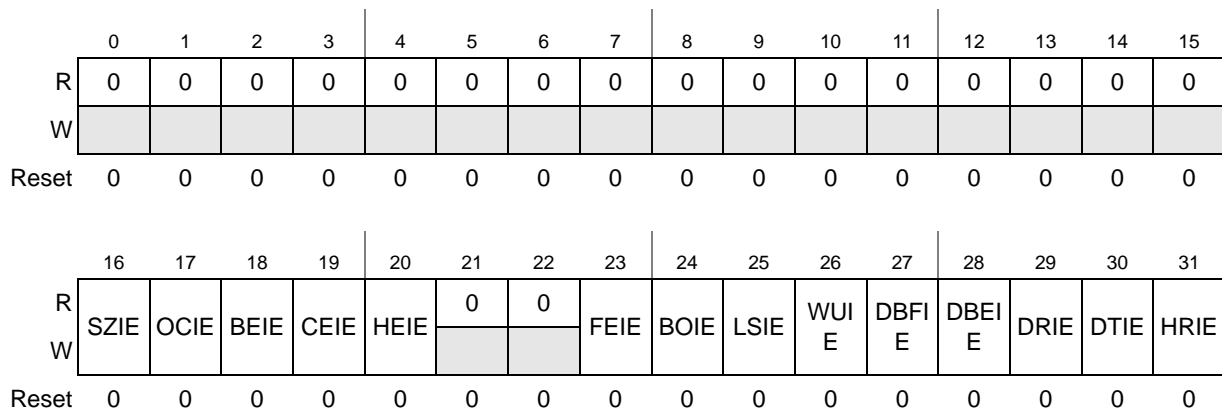
**Table 207. Operating mode selection**

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

**23.7.1.2 LIN interrupt enable register (LINIER)**

Offset: 0x0004

Access: User read/write



**Figure 207. LIN interrupt enable register (LINIER)**

**Table 208. LINIER field descriptions**

Field	Description
SZIE	Stuck at Zero Interrupt Enable 0 No interrupt when SZF bit in LINESR or UARTSR is set. 1 Interrupt generated when SZF bit in LINESR or UARTSR is set.
OCIE	Output Compare Interrupt Enable 0 No interrupt when OCF bit in LINESR or UARTSR is set. 1 Interrupt generated when OCF bit in LINESR or UARTSR is set.
BEIE	Bit Error Interrupt Enable 0 No interrupt when BEF bit in LINESR is set. 1 Interrupt generated when BEF bit in LINESR is set.





Table 208. LINIER field descriptions (continued)

Field	Description
CEIE	Checksum Error Interrupt Enable 0 No interrupt on Checksum error. 1 Interrupt generated when checksum error flag (CEF) in LINESR is set.
HEIE	Header Error Interrupt Enable 0 No interrupt on Break Delimiter error, Synch Field error, Identifier field error. 1 Interrupt generated on Break Delimiter error, Synch Field error, Identifier field error.
FEIE	Framing Error Interrupt Enable 0 No interrupt on Framing error. 1 Interrupt generated on Framing error.
BOIE	Buffer Overrun Interrupt Enable 0 No interrupt on Buffer overrun. 1 Interrupt generated on Buffer overrun.
LSIE	LIN State Interrupt Enable 0 No interrupt on LIN state change. 1 Interrupt generated on LIN state change. This interrupt can be used for debugging purposes. It has no status flag but is reset when writing '1111' into LINS[0:3] in the LINSR.
WUIE	Wake-up Interrupt Enable 0 No interrupt when WUF bit in LINSR or UARTSR is set. 1 Interrupt generated when WUF bit in LINSR or UARTSR is set.
DBFIE	Data Buffer Full Interrupt Enable 0 No interrupt when buffer data register is full. 1 Interrupt generated when data buffer register is full.
DBEIE	Data Buffer Empty Interrupt Enable 0 No interrupt when buffer data register is empty. 1 Interrupt generated when data buffer register is empty.
DRIE	Data Reception Complete Interrupt Enable 0 No interrupt when data reception is completed. 1 Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set.
DTIE	Data Transmitted Interrupt Enable 0 No interrupt when data transmission is completed. 1 Interrupt generated when data transmitted flag (DTF) is set in LINSR or UARTSR.
HRIE	Header Received Interrupt Enable 0 No interrupt when a valid LIN header has been received. 1 Interrupt generated when a valid LIN header has been received, that is, HRF bit in LINSR is set.

23.7.1.3 LIN status register (LINSR)

Offset: 0x0008

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LINS				0	0	RMB	0	RBS Y	RPS	WUF	DBF F	DBE F	DRF	DTF	HRF
W	w1c						w1c		w1c		w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 208. LIN status register (LINSR)

Table 209. LINSR field descriptions

Field	Description
LINS	<p>LIN modes / normal mode states</p> <p><b>0000: Sleep mode</b> LINFlex is in Sleep mode to save power consumption.</p> <p><b>0001: Initialization mode</b> LINFlex is in Initialization mode.</p> <p>Normal mode states</p> <p><b>0010: Idle</b> This state is entered on several events: – SLEEP bit and INIT bit in LINC1R1 have been cleared by software, – A falling edge has been received on RX pin and AWUM bit is set, – The previous frame reception or transmission has been completed or <b>aborted</b>.</p> <p><b>0011: Break</b> In Slave mode, a falling edge followed by a dominant state has been detected. Receiving Break. <b>Note:</b> In Slave mode, in case of error new LIN state can be either Idle or Break depending on last bit state. If last bit is dominant new LIN state is Break, otherwise Idle. In Master mode, Break transmission ongoing.</p> <p><b>0100: Break Delimiter</b> In Slave mode, a valid Break has been detected. See <a href="#">Section 23.7.1.1: LIN control register 1 (LINC1R1)</a> for break length configuration (10-bit or 11-bit). Waiting for a rising edge. In Master mode, Break transmission has been completed. Break Delimiter transmission is ongoing.</p> <p><b>0101: Synch Field</b> In Slave mode, a valid Break Delimiter has been detected (recessive state for at least one bit time). Receiving Synch Field. In Master mode, Synch Field transmission is ongoing.</p> <p><b>0110: Identifier Field</b> In Slave mode, a valid Synch Field has been received. Receiving Identifier Field. In Master mode, identifier transmission is ongoing.</p> <p><b>0111: Header reception/transmission completed</b> In Slave mode, a valid header has been received and identifier field is available in the BIDR. In Master mode, header transmission is completed.</p> <p><b>1000: Data reception/transmission</b> Response reception/transmission is ongoing.</p> <p><b>1001: Checksum</b> Data reception/transmission completed. Checksum reception/transmission ongoing. In UART mode, only the following states are flagged by the LIN state bits: – Init – Sleep – Idle – Data transmission/reception</p>
RMB	<p>Release Message Buffer</p> <p>0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.</p>

**Table 209. LINSR field descriptions (continued)**

Field	Description
RBSY	Receiver Busy Flag 0 Receiver is idle 1 Reception ongoing <b>Note:</b> In Slave mode, after header reception, if BIDR[DIR] = 0 and reception starts then this bit is set. In this case, user cannot program LINCR2[DTRQ] = 1.
RPS	LIN receive pin state This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin when: – Slave is in Sleep mode – Master is in Sleep mode or idle state This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if WUIE bit in LINIER is set.
DBFF	Data Buffer Full Flag This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL > 7). This bit must be cleared by software. It is reset by hardware in Initialization mode.
DBEF	Data Buffer Empty Flag This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL > 7). This bit must be cleared by software, once buffer has been filled again, in order to start transmission. This bit is reset by hardware in Initialization mode.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. <b>Note:</b> This flag is not set in case of bit error or framing error.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. <b>Note:</b> This flag is not set in case of bit error if IOBE bit is reset.
HRF	Header Reception Flag This bit is set by hardware and indicates a valid header reception is completed. This bit must be cleared by software. This bit is reset by hardware in Initialization mode and at end of completed or aborted frame. <b>Note:</b> If filters are enabled, this bit is set only when identifier software filtering is required, that is to say: – All filters are inactive and BF bit in LINCR1 is set – No match in any filter and BF bit in LINCR1 is set – TX filter match

23.7.1.4 LIN error status register (LINESR)

Offset: 0x000C

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	BEF	CEF	SFE F	BDE F	IDPEF	FEF	BOF	0	0	0	0	0	0	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 209. LIN error status register (LINESR)

Table 210. LINESR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. If the dominant state continues, SZF flag is set again after 87-bit time. It is cleared by software.
OCF	Output Compare Flag 0 No output compare event occurred 1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. If this bit is set and IOT bit in LINTCSR is set, LINFlex moves to Idle state. If LTOM bit in LINTCSR is set, then OCF is cleared by hardware in Initialization mode. If LTOM bit is cleared, then OCF maintains its status whatever the mode is.
BEF	Bit Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a bit error. This error can occur during response field transmission (Slave and Master modes) or during header transmission (in Master mode). This bit is cleared by software.
CEF	Checksum Error Flag This bit is set by hardware and indicates that the received checksum does not match the hardware calculated checksum. This bit is cleared by software. <b>Note:</b> This bit is never set if CCD or CFD bit in LINCR1 is set.
SFEF	Synch Field Error Flag This bit is set by hardware and indicates that a Synch Field error occurred (inconsistent Synch Field).
BDEF	Break Delimiter Error Flag This bit is set by hardware and indicates that the received Break Delimiter is too short (less than one bit time).

**Table 210. LINESR field descriptions (continued)**

Field	Description
IDPEF	Identifier Parity Error Flag This bit is set by hardware and indicates that a Identifier Parity error occurred. <b>Note:</b> Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit). This error can occur during reception of any data in the response field (Master or Slave mode) or during reception of Synch Field or Identifier Field in Slave mode.
BOF	Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINC1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites the buffer. It can be cleared by software.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

**23.7.1.5 UART mode control register (UARTCR)**

Offset: 0x0010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	TDFL		0	RDFL		0	0	0	0	RXEN	TXE N	OP	PCE	WL	UAR T
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 210. UART mode control register (UARTCR)**

**Table 211. UARTCR field descriptions**

Field	Description
TDFL	Transmitter Data Field length This field sets the number of bytes to be transmitted in UART mode. It can be programmed only when the UART bit is set. TDFL[0:1] = Transmit buffer size – 1. 00 Transmit buffer size = 1. 01 Transmit buffer size = 2. 10 Transmit buffer size = 3. 11 Transmit buffer size = 4.

**Table 211. UARTCR field descriptions (continued)**

Field	Description
RDFL	Receiver Data Field length This field sets the number of bytes to be received in UART mode. It can be programmed only when the UART bit is set. RDFL[0:1] = Receive buffer size – 1. 00 Receive buffer size = 1. 01 Receive buffer size = 2. 10 Receive buffer size = 3. 11 Receive buffer size = 4.
RXEN	Receiver Enable 0 Receiver disable. 1 Receiver enable. This bit can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0 Transmitter disable. 1 Transmitter enable. This bit can be programmed only when the UART bit is set. <b>Note:</b> Transmission starts when this bit is set and when writing DATA0 in the BDRL register.
OP	Odd Parity 0 Sent parity is even. 1 Sent parity is odd. This bit can be programmed in Initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0 Parity transmit/check disable. 1 Parity transmit/check enable. This bit can be programmed in Initialization mode only when the UART bit is set.
WL	Word Length in UART mode 0 7-bit data + parity bit. 1 8-bit data (or 9-bit if PCE is set). This bit can be programmed in Initialization mode only when the UART bit is set.
UART	UART mode enable 0 LIN mode. 1 UART mode. This bit can be programmed in Initialization mode only.

23.7.1.6 UART mode status register (UARTSR)

Offset: 0x0014

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	PE3	PE2	PE1	PE0	RMB	FEF	BOF	RPS	WUF	0	0	DRF	DTF	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 211. UART mode status register (UARTSR)

Table 212. UARTSR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.
OCF	OCF Output Compare Flag 0 No output compare event occurred. 1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCCR. An interrupt is generated if the OCIE bit in LINIER register is set.
PE3	Parity Error Flag Rx3 This bit indicates if there is a parity error in the corresponding received byte (Rx3). See <a href="#">Section 23.8.1.1: Buffer in UART mode</a> . No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE2	Parity Error Flag Rx2 This bit indicates if there is a parity error in the corresponding received byte (Rx2). See <a href="#">Section 23.8.1.1: Buffer in UART mode</a> . No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE1	Parity Error Flag Rx1 This bit indicates if there is a parity error in the corresponding received byte (Rx1). See <a href="#">Section 23.8.1.1: Buffer in UART mode</a> . No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE0	Parity Error Flag Rx0 This bit indicates if there is a parity error in the corresponding received byte (Rx0). See <a href="#">Section 23.8.1.1: Buffer in UART mode</a> . No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.



Table 212. UARTSR field descriptions (continued)

Field	Description
RMB	<p>Release Message Buffer</p> <p>0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer.</p> <p>This bit is cleared by hardware in Initialization mode.</p>
FEF	<p>Framing Error Flag</p> <p>This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit).</p>
BOF	<p>Buffer Overrun Flag</p> <p>This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites buffer. it can be cleared by software.</p>
RPS	<p>LIN Receive Pin State</p> <p>This bit reflects the current status of LINRX pin for diagnostic purposes.</p>
WUF	<p>Wake-up Flag</p> <p>This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin in Sleep mode.</p> <p>This bit must be cleared by software. It is reset by hardware in Initialization mode.</p> <p>An interrupt is generated if WUIE bit in LINIER is set.</p>
DRF	<p>Data Reception Completed Flag</p> <p>This bit is set by hardware and indicates the data reception is completed, that is, the number of bytes programmed in RDFL[0:1] in UARTCR have been received.</p> <p>This bit must be cleared by software.</p> <p>It is reset by hardware in Initialization mode.</p> <p>An interrupt is generated if DRIE bit in LINIER is set.</p> <p><b>Note:</b> In UART mode, this flag is set in case of framing error, parity error or overrun.</p>
DTF	<p>Data Transmission Completed Flag</p> <p>This bit is set by hardware and indicates the data transmission is completed, that is, the number of bytes programmed in TDFL[0:1] have been transmitted.</p> <p>This bit must be cleared by software.</p> <p>It is reset by hardware in Initialization mode.</p> <p>An interrupt is generated if DTIE bit in LINIER is set.</p>
NF	<p>Noise Flag</p> <p>This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.</p>

**23.7.1.7 LIN timeout control status register (LINTCSR)**

Offset: 0x0018

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	LTOM	IOT	TOCE	CNT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

**Figure 212. LIN timeout control status register (LINTCSR)**

**Table 213. LINTCSR field descriptions**

Field	Description
LTOM	LIN timeout mode 0 LIN timeout mode (header, response and frame timeout detection). 1 Output compare mode. This bit can be set/cleared in Initialization mode only.
IOT	Idle on Timeout 0 LIN state machine not reset to Idle on timeout event. 1 LIN state machine reset to Idle on timeout event. This bit can be set/cleared in Initialization mode only.
TOCE	Timeout counter enable 0 Timeout counter disable. OCF bit in LINESR or UARTSR is not set on an output compare event. 1 Timeout counter enable. OCF bit is set if an output compare event occurs. TOCE bit is configurable by software in Initialization mode. If LIN state is not Init and if timer is in LIN timeout mode, then hardware takes control of TOCE bit.
CNT	Counter Value This field indicates the LIN timeout counter value.

**23.7.1.8 LIN output compare register (LINOOCR)**

Offset: 0x001C Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OC2 <sup>(1)</sup>								OC1 <sup>(1)</sup>							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 213. LIN output compare register (LINOOCR)**

1. If LINTCSR[LTOM] = 0, this field is read-only.

**Table 214. LINOOCR field descriptions**

Field	Description
OC2	Output compare 2 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.
OC1	Output compare 1 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.

**23.7.1.9 LIN timeout control register (LINTOCR)**

Offset: 0x0020 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	0	0	0	0	RTO				0	HTO							
W																	
Reset	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	0	

**Figure 214. LIN timeout control register (LINTOCR)**

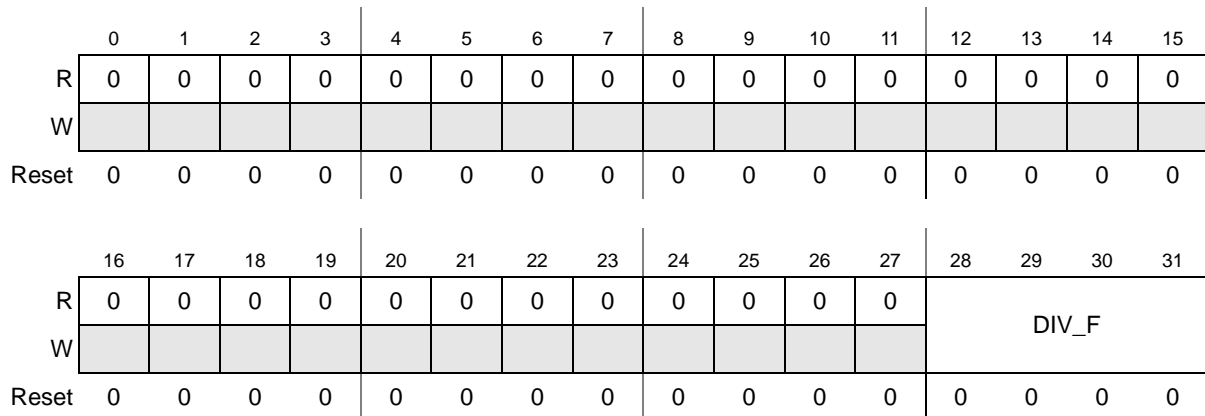
**Table 215. LINTOCR field descriptions**

Field	Description
RTO	Response timeout value This field contains the response timeout duration (in bit time) for 1 byte. The reset value is 0xE = 14, corresponding to $T_{Response\_Maximum} = 1.4 \times T_{Response\_Nominal}$
HTO	Header timeout value This field contains the header timeout duration (in bit time). This value does not include the Break and the Break Delimiter. The reset value is the 0x2C = 44, corresponding to $T_{Header\_Maximum}$ . Programming LINSR[MME] = 1 changes the HTO value to 0x1C = 28. This field can be written only in Slave mode.

**23.7.1.10 LIN fractional baud rate register (LINFBR)**

Offset: 0x0024

Access: User read/write



**Figure 215. LIN fractional baud rate register (LINFBR)**

**Table 216. LINFBR field descriptions**

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlex divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F / 16. This field can be written in Initialization mode only.

23.7.1.11 LIN integer baud rate register (LINIBRR)

Offset: 0x0028

Access: User read/write

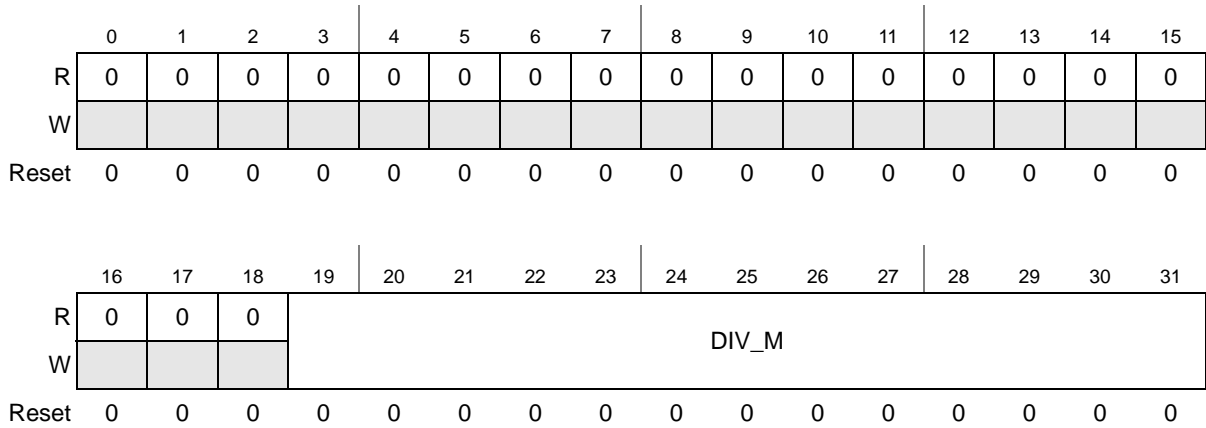


Figure 216. LIN integer baud rate register (LINIBRR)

Table 217. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa This field defines the LINFlex divider (LFDIV) mantissa value (see <a href="#">Table 218</a> ). This field can be written in Initialization mode only.

Table 218. Integer baud rate selection

DIV_M[0:12]	Mantissa
0x0000	LIN clock disabled
0x0001	1
...	...
0x1FFE	8190
0x1FFF	8191

23.7.1.12 LIN checksum field register (LINCFR)

Offset: 0x002C Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CF							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 217. LIN checksum field register (LINCFR)

Table 219. LINCFR field descriptions

Field	Description
CF	Checksum bits When LINCR1[CCD] = 0, this field is read-only. When LINCR1[CCD] = 1, this field is read/write. See <a href="#">Table 205</a> .

23.7.1.13 LIN control register 2 (LINCR2)

Offset: 0x0030 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0			0	0	0	0	0	0	0	0	0	0	0	0	0
W		IOBE	IOPE	WUR	DDR	DTR	ABR	HTR								
				Q	Q	Q	Q	Q								
Reset	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 218. LIN control register 2 (LINCR2)

Table 220. LINCR2 field descriptions

Field	Description
IOBE	<p>Idle on Bit Error</p> <p>0 Bit error does not reset LIN state machine. 1 Bit error reset LIN state machine.</p> <p>This bit can be set/cleared in Initialization mode only.</p>
IOPE	<p>Idle on Identifier Parity Error</p> <p>0 Identifier Parity error does not reset LIN state machine. 1 Identifier Parity error reset LIN state machine.</p> <p>This bit can be set/cleared in Initialization mode only.</p>
WURQ	<p>Wake-up Generation Request</p> <p>Setting this bit generates a wake-up pulse. It is reset by hardware when the wake-up character has been transmitted. The character sent is copied from DATA0 in BDRL buffer. Note that this bit cannot be set in Sleep mode. Software has to exit Sleep mode before requesting a wake-up. Bit error is not checked when transmitting the wake-up request.</p>
DDRQ	<p>Data Discard Request</p> <p>Set by software to stop data reception if the frame does not concern the node. This bit is reset by hardware once LINFlex has moved to idle state. In Slave mode, this bit can be set only when HRF bit in LINSR is set and identifier did not match any filter.</p>
DTRQ	<p>Data Transmission Request</p> <p>Set by software in Slave mode to request the transmission of the LIN Data field stored in the Buffer data register. This bit can be set only when HRF bit in LINSR is set.</p> <p>Cleared by hardware when the request has been completed or aborted or on an error condition.</p> <p>In Master mode, this bit is set by hardware when BIDR[DIR] = 1 and header transmission is completed.</p>
ABRQ	<p>Abort Request</p> <p>Set by software to abort the current transmission.</p> <p>Cleared by hardware when the transmission has been aborted. LINFlex aborts the transmission at the end of the current bit.</p> <p>This bit can also abort a wake-up request.</p> <p>It can also be used in UART mode.</p>
HTRQ	<p>Header Transmission Request</p> <p>Set by software to request the transmission of the LIN header.</p> <p>Cleared by hardware when the request has been completed or aborted.</p> <p>This bit has no effect in UART mode.</p>

23.7.1.14 Buffer identifier register (BIDR)

Offset: 0x0034

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DFL				DIR		CCS		0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 219. Buffer identifier register (BIDR)

Table 221. BIDR field descriptions

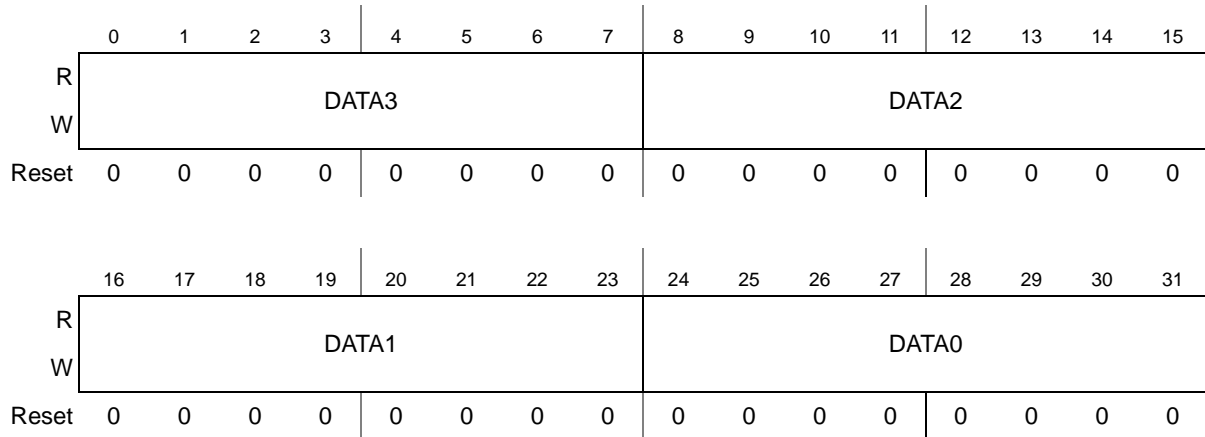
Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1. Normally, LIN uses only DFL[2:0] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[2:0] only. DFL[5:3] are provided to manage extended frames.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDR registers. 1 LINFlex transmits the data from the BDR registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier. In LIN slave mode (MME bit cleared in LINCR1), this bit must be configured before the header reception. If the slave has to manage frames with 2 types of checksum, filters must be configured.
ID	Identifier Identifier part of the identifier field without the identifier parity.



**23.7.1.15 Buffer data register LSB (BDRL)**

Offset: 0x0038

Access: User read/write

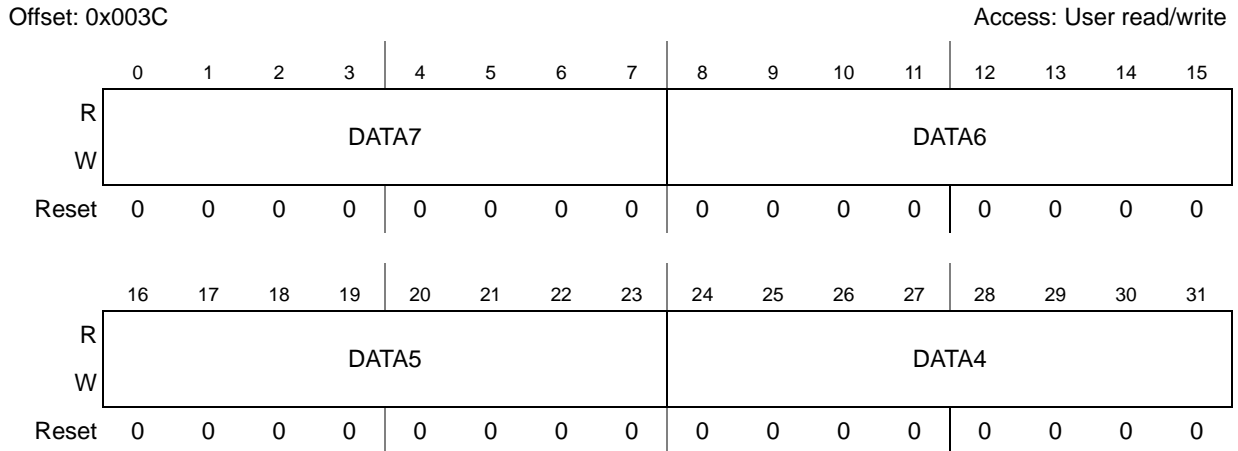


**Figure 220. Buffer data register LSB (BDRL)**

**Table 222. BDRL field descriptions**

Field	Description
DATA3	Data Byte 3 Data byte 3 of the data field.
DATA2	Data Byte 2 Data byte 2 of the data field.
DATA1	Data Byte 1 Data byte 1 of the data field.
DATA0	Data Byte 0 Data byte 0 of the data field.

**23.7.1.16 Buffer data register MSB (BDRM)**

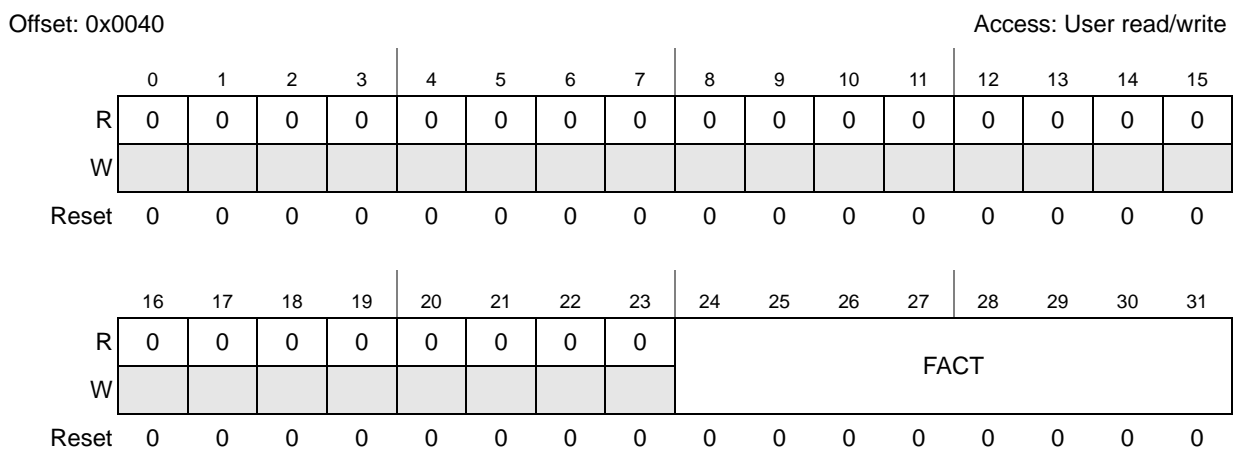


**Figure 221. Buffer data register MSB (BDRM)**

**Table 223. BDRM field descriptions**

Field	Description
DATA7	Data Byte 7 Data byte 7 of the data field.
DATA6	Data Byte 6 Data byte 6 of the data field.
DATA5	Data Byte 5 Data byte 5 of the data field.
DATA4	Data Byte 4 Data byte 4 of the data field.

**23.7.1.17 Identifier filter enable register (IFER)**



**Figure 222. Identifier filter enable register (IFER)**

Table 224. IFER field descriptions

Field	Description
FACT	Filter activation (see <a href="#">Table 225</a> ) 0 Filters $2n$ and $2n + 1$ are deactivated. 1 Filters $2n$ and $2n + 1$ are activated. This field can be set/cleared in Initialization mode only.

Table 225. IFER[FACT] configuration

Bit	Value	Result
FACT[0]	0	Filters 0 and 1 are deactivated.
	1	Filters 0 and 1 are activated.
FACT[1]	0	Filters 2 and 3 are deactivated.
	1	Filters 2 and 3 are activated.
FACT[2]	0	Filters 4 and 5 are deactivated.
	1	Filters 4 and 5 are activated.
FACT[3]	0	Filters 6 and 7 are deactivated.
	1	Filters 6 and 7 are activated.
FACT[4]	0	Filters 8 and 9 are deactivated.
	1	Filters 8 and 9 are activated.
FACT[5]	0	Filters 10 and 11 are deactivated.
	1	Filters 10 and 11 are activated.
FACT[6]	0	Filters 12 and 13 are deactivated.
	1	Filters 12 and 13 are activated.
FACT[7]	0	Filters 14 and 15 are deactivated.
	1	Filters 14 and 15 are activated.

**23.7.1.18 Identifier filter match index (IFMI)**

Address: Base + 0x0044

Access: User read-only

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0		IFMI[0:4]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 223. Identifier filter match index (IFMI)**

**Table 226. IFMI field descriptions**

Field	Description
0:26	Reserved
IFMI[0:4] 27:31	Filter match index This register contains the index corresponding to the received identifier. It can be used to directly write or read the data in SRAM (see <a href="#">Section 23.8.2.2: Slave mode</a> for more details). When no filter matches, IFMI[0:4] = 0. When Filter <i>n</i> is matching, IFMI[0:4] = <i>n</i> + 1.

**23.7.1.19 Identifier filter mode register (IFMR)**

Offset: 0x0048

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	IFM							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 224. Identifier filter mode register (IFMR)**

Table 227. IFMR field descriptions

Field	Description
IFM	Filter mode (see <a href="#">Table 228</a> ). 0 Filters $2n$ and $2n + 1$ are in identifier list mode. 1 Filters $2n$ and $2n + 1$ are in mask mode (filter $2n + 1$ is the mask for the filter $2n$ ).

Table 228. IFMR[IFM] configuration

Bit	Value	Result
IFM[0]	0	Filters 0 and 1 are in identifier list mode.
	1	Filters 0 and 1 are in mask mode (filter 1 is the mask for the filter 0).
IFM[1]	0	Filters 2 and 3 are in identifier list mode.
	1	Filters 2 and 3 are in mask mode (filter 3 is the mask for the filter 2).
IFM[2]	0	Filters 4 and 5 are in identifier list mode.
	1	Filters 4 and 5 are in mask mode (filter 5 is the mask for the filter 4).
IFM[3]	0	Filters 6 and 7 are in identifier list mode.
	1	Filters 6 and 7 are in mask mode (filter 7 is the mask for the filter 6).
IFM[4]	0	Filters 8 and 9 are in identifier list mode.
	1	Filters 8 and 9 are in mask mode (filter 9 is the mask for the filter 8).
IFM[5]	0	Filters 10 and 11 are in identifier list mode.
	1	Filters 10 and 11 are in mask mode (filter 11 is the mask for the filter 10).
IFM[6]	0	Filters 12 and 13 are in identifier list mode.
	1	Filters 12 and 13 are in mask mode (filter 13 is the mask for the filter 12).
IFM[7]	0	Filters 14 and 15 are in identifier list mode.
	1	Filters 14 and 15 are in mask mode (filter 15 is the mask for the filter 14).

**23.7.1.20 Identifier filter control register (IFCR2n)**

Offsets : 0x004C–0x0084 (8 registers)

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0		DFL			DIR	CCS	0	0	ID				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 225. Identifier filter control register (IFCR2n)**

*Note:* This register can be written in Initialization mode only.

**Table 229. IFCR2n field descriptions**

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex transmits the data from the BDRL and BDRM registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity.

**23.7.1.21 Identifier filter control register (IFCR2n + 1)**

Offsets: 0x0050–0x0088 (8 registers)

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0		DFL			DIR	CCS	0	0	ID				
W					DFL			DIR	CCS			w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 226. Identifier filter control register (IFCR2n + 1)**

Note: This register can be written in Initialization mode only.

**Table 230. IFCR2n + 1 field descriptions**

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex transmits the data from the BDRL and BDRM registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data field only. This is compatible with LIN specification 1.3 and earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity

**23.8 Functional description**

**23.8.1 UART mode**

The main features in the UART mode are

- Full duplex communication
- 8- or 9-bit data with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management



**8-bit data frames:** The 8th bit can be a data or a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 7 data bits is 1. An odd parity is cleared in this case.

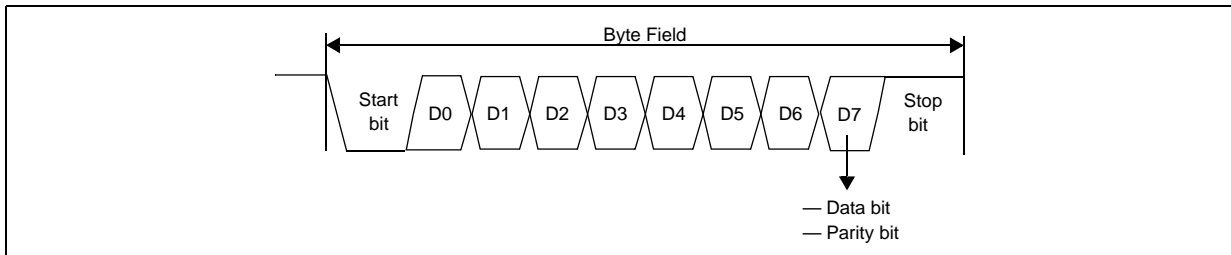


Figure 227. UART mode 8-bit data frame

**9-bit frames:** The 9th bit is a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 8 data bits is 1. An odd parity is cleared in this case.

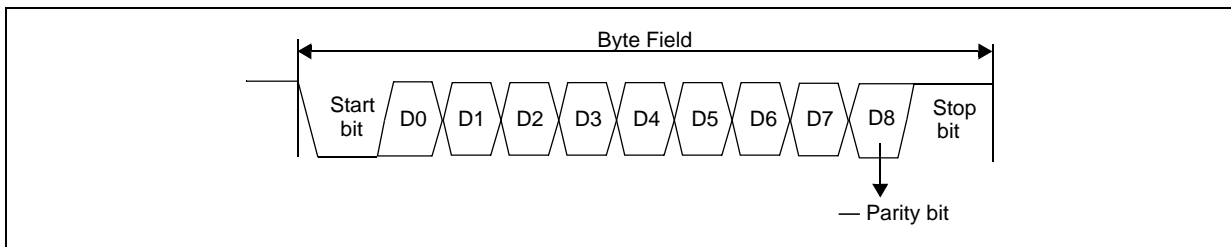


Figure 228. UART mode 9-bit data frame

23.8.1.1 Buffer in UART mode

The 8-byte buffer is divided into two parts: one for receiver and one for transmitter as shown in [Table 231](#).

Table 231. Message buffer

Buffer data register	LIN mode		UART mode	
BDRL[0:31]	Transmit/Receive buffer	DATA0[0:7]	Transmit buffer	Tx0
		DATA1[0:7]		Tx1
		DATA2[0:7]		Tx2
		DATA3[0:7]		Tx3
BDRM[0:31]		DATA4[0:7]	Receive buffer	Rx0
		DATA5[0:7]		Rx1
		DATA6[0:7]		Rx2
		DATA7[0:7]		Rx3



### 23.8.1.2 UART transmitter

In order to start transmission in UART mode, you must program the UART bit and the transmitter enable (TXEN) bit in the UARTCR to 1. Transmission starts when DATA0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by UARTCR[TDFL] (see [Table 211](#)).

The Transmit buffer is 4 bytes, hence a 4-byte maximum transmission can be triggered. Once the programmed number of bytes has been transmitted, the UARTSR[DTF] bit is set. If UARTCR[TXEN] is reset during a transmission then the current transmission is completed and no further transmission can be invoked.

### 23.8.1.3 UART receiver

The UART receiver is active as soon as the user exits Initialization mode and programs UARTCR[RXEN] = 1. There is a dedicated 4-byte data buffer for received data bytes. Once the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRF] bit is set. If the RXEN bit is reset during a reception then the current reception is completed and no further reception can be invoked until RXEN is set.

If a parity error occurs during reception of any byte, then the corresponding PEx bit in the UARTSR is set. No interrupt is generated in this case. If a framing error occurs in any byte (UARTSR[FE] = 1) then an interrupt is generated if the LINIER[FEIE] bit is set.

If the last received frame has not been read from the buffer (that is, RMB bit is not reset by the user) then upon reception of the next byte an overrun error occurs (UARTSR[BOF] = 1) and one message will be lost. Which message is lost depends on the configuration of LINCR1[RBLM].

- If the buffer lock function is disabled (LINCR1[RBLM] = 0) the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled (LINCR1[RBLM] = 1) the most recent message is discarded and the previous message is available in the buffer.

An interrupt is generated if the LINIER[BOIE] bit is set.

### 23.8.1.4 Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC\_ME). In UART mode, the LINFlex controller acknowledges a clock gating request once the data transmission and data reception are completed, that is, once the Transmit buffer is empty and the Receive buffer is full.

## 23.8.2 LIN mode

LIN mode comprises four submodes:

- Master mode
- Slave mode
- Slave mode with identifier filtering
- Slave mode with automatic resynchronization

These submodes are described in the following pages.

### 23.8.2.1 Master mode

In Master mode the application uses the message buffer to handle the LIN messages. Master mode is selected when the LINCR1[MME] bit is set.

#### 23.8.2.1.1 LIN header transmission

According to the LIN protocol any communication on the LIN bus is triggered by the Master sending a header. The header is transmitted by the Master task while the data is transmitted by the Slave task of a node.

To transmit a header with LINFlex the application must set up the identifier, the data field length and configure the message (direction and checksum type) in the BIDR before requesting the header transmission by setting LINCR2[HTRQ].

#### 23.8.2.1.2 Data transmission (transceiver as publisher)

When the master node is publisher of the data corresponding to the identifier sent in the header, then the slave task of the master has to send the data in the Response part of the LIN frame. Therefore, the application must provide the data to LINFlex before requesting the header transmission. The application stores the data in the message buffer BDR. According to the data field length, LINFlex transmits the data and the checksum. The application uses the BDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

If the response has been sent successfully, the LINSR[DTF] bit is set. In case of error, the DTF flag is not set and the corresponding error flag is set in the LINESR (see [Section 23.8.2.1.6: Error handling](#)).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBEF] bit is set after the first 8 bytes have been transmitted. The application has to update the buffer BDR before resetting the DBEF bit. The transmission of the next bytes starts when the DBEF bit is reset.

After the last data byte (or the checksum byte) has been sent, the DTF flag is set.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the application sets this bit the response is sent by LINFlex (publisher). Resetting this bit configures the message buffer as subscriber.

#### 23.8.2.1.3 Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlex stores the data received from the slave in the message buffer and stores the message status in the LINSR.

If the response has been received successfully, the LINSR[DRF] is set. In case of error, the DRF flag is not set and the corresponding error flag is set in the LINESR (see [Section 23.8.2.1.6: Error handling](#)).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBFF] bit is set once the first 8 bytes have been received. The application has to read the buffer BDR before resetting the DBFF bit. Once the last data byte (or the checksum byte) has been received, the DRF flag is set.

#### 23.8.2.1.4 Data discard

To discard data from a slave, the BIDR[DIR] bit must be reset and the LINCR2[DDRQ] bit must be set before starting the header transmission.

#### 23.8.2.1.5 Error detection

LINFlex is able to detect and handle LIN communication errors. A code stored in the LIN error status register (LINESR) signals the errors to the software.

In Master mode, the following errors are detected:

- **Bit error:** During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error:** A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error:** The computed checksum does not match the received one.
- **Response and Frame timeout:** See [Section 23.8.3: 8-bit timeout counter](#), for more details.

#### 23.8.2.1.6 Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if LINIER[BEIE] = 1.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

#### 23.8.2.1.7 Overrun

Once the message buffer is full, the next valid message reception leads to an overrun and a message is lost. The hardware sets the BOF bit in the LINSR to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINCR1[RBLM] = 0) the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled (LINCR1[RBLM] = 1) the most recent message is discarded and the previous message is available in the buffer.

#### 23.8.2.2 Slave mode

In Slave mode the application uses the message buffer to handle the LIN messages. Slave mode is selected when LINCR1[MME] = 0.

##### 23.8.2.2.1 Data transmission (transceiver as publisher)

When LINFlex receives the identifier, the LINSR[HRF] is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR, fill the BDR registers, specify the data field length using the BIDR[DFL] and trigger the data transmission by setting the LINCR2[DTRQ] bit.

One or several identifier filters can be configured for transmission by setting the IFCR<sub>x</sub>[DIR] bit and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in transmission and activated, and if the received identifier matches the filter, a specific TX interrupt (instead of an RX interrupt) is generated.

Typically, the application has to copy the data from SRAM locations to the BDAR. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data to the BDAR (see [Figure 230](#)).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BIDR. The software fills the BDAR and triggers the data transmission by programming LINCR2[DTRQ] = 1.

If LINFlex cannot provide enough TX identifier filters to handle all identifiers the software has to transmit data for, then a filter can be configured in mask mode (see [Section 23.8.2.3: Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

#### 23.8.2.2.2 Data reception (transceiver as subscriber)

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR and specify the data field length using the BIDR[DFL] field before receiving the stop bit of the first byte of data field.

When the checksum reception is completed, an RX interrupt is generated to allow the software to read the received data in the BDR registers.

One or several identifier filters can be configured for reception by programming IFCR<sub>x</sub>[DIR] = 0 and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in reception and activated, and if the received identifier matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the application has to copy the data from the BDAR to SRAM locations. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data from the BDAR to the SRAM (see [Figure 230](#)).

Using a filter avoids the software reading the ID value in the BIDR, and configuring the direction, the data field length and the checksum type in the BIDR.

If LINFlex cannot provide enough RX identifier filters to handle all identifiers the software has to receive the data for, then a filter can be configured in mask mode (see [Section 23.8.2.3: Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

### 23.8.2.2.3 Data discard

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. If the received identifier does not concern the node, you must program LINCRC2[DDRQ] = 1. LINFlex returns to idle state after bit DDRQ is set.

### 23.8.2.2.4 Error detection

In Slave mode, the following errors are detected:

- **Header error:** An error occurred during header reception (Break Delimiter error, Inconsistent Synch Field, Header Timeout).
- **Bit error:** During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error:** A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error:** The computed checksum does not match the received one.

### 23.8.2.2.5 Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if the BEIE bit in the LINIER is set.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

During header reception, a Break Delimiter error, an Inconsistent Synch Field or a Timeout error leads LINFlex to discard the header. An interrupt is generated if LINIER[HEIE] = 1. LINFlex returns to idle state.

### 23.8.2.2.6 Valid header

A received header is considered as valid when it has been received correctly according to the LIN protocol.

If a valid Break Field and Break Delimiter come before the end of the current header or at any time during a data field, the current header or data is discarded and the state machine synchronizes on this new break.

### 23.8.2.2.7 Valid message

A received or transmitted message is considered as valid when the data has been received or transmitted without error according to the LIN protocol.

### 23.8.2.2.8 Overrun

Once the message buffer is full, the next valid message reception leads to an overrun and a message is lost. The hardware sets the BOF bit in the LINSR to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINC1[RBLM] = 0) the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled (LINC1[RBLM] = 1) the most recent message is discarded and the previous message is available in the buffer.

### 23.8.2.3 Slave mode with identifier filtering

In the LIN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On header reception a slave node decides—depending on the identifier value—whether the software needs to receive or send a response. If the message does not target the node, it must be discarded without software intervention.

To fulfill this requirement, the LINFlex controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources that would otherwise be needed by software for filtering.

#### 23.8.2.3.1 Filter mode

Usually each of the eight IFCR registers filters one dedicated identifier, but this limits the number of identifiers LINFlex can handle to the number of IFCR registers implemented in the device. Therefore, in order to be able to handle more identifiers, the filters can be configured in mask mode.

In **identifier list mode** (the default mode), both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register.

In **mask mode**, the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”. For the bit mapping and registers organization, please see [Figure 229](#).

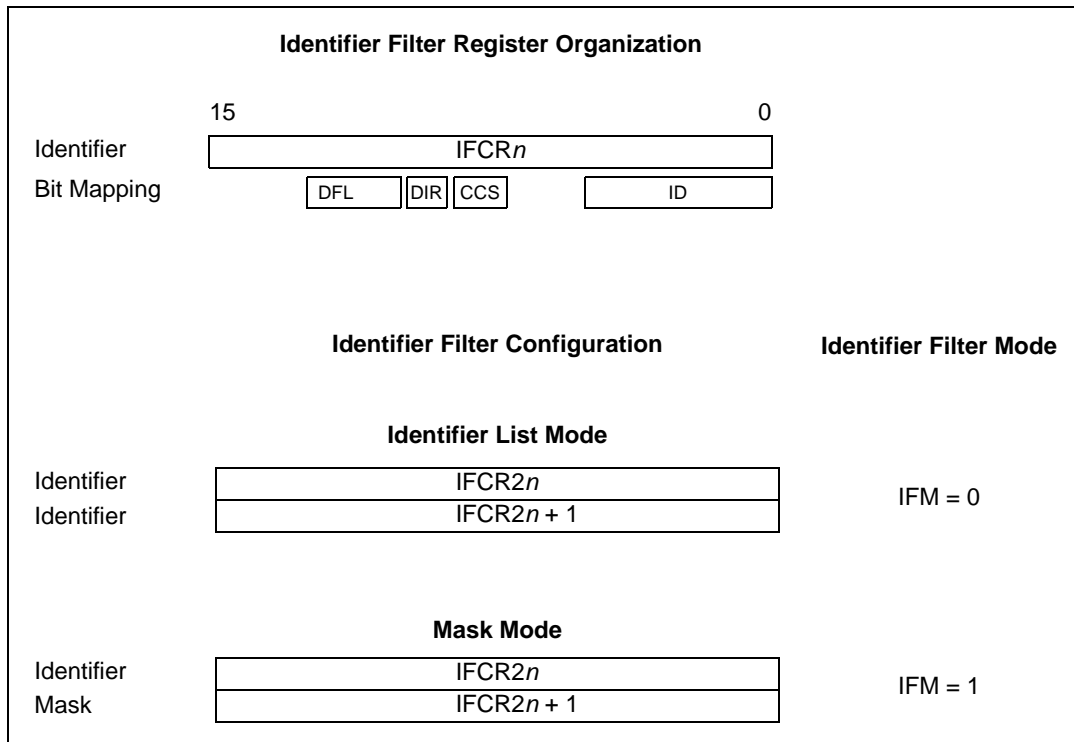


Figure 229. Filter configuration—register organization

### 23.8.2.3.2 Identifier filter mode configuration

The identifier filters are configured in the IFCR<sub>x</sub> registers. To configure an identifier filter the filter must first be activated by programming IFER[FACT] = 1. The **identifier list** or **identifier mask** mode for the corresponding IFCR<sub>x</sub> registers is configured by the IFMR[IFM] bit. For each filter, the IFCR<sub>x</sub> register configures the ID (or the mask), the direction (TX or RX), the data field length, and the checksum type.

If no filter is active, an RX interrupt is generated on any received identifier event.

If at least one active filter is configured as TX, all received identifiers matching this filter generate a TX interrupt.

If at least one active filter is configured as RX, all received identifiers matching this filter generate an RX interrupt.

If no active filter is configured as RX, all received identifiers not matching TX filter(s) generate an RX interrupt.

Table 232. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as TX	Number of active filters configured as RX	Interrupt vector
0	0	0	RX interrupt on all identifiers
a (a > 0)	a	0	— TX interrupt on identifiers matching the filters, — RX interrupt on all other identifiers if BF bit is set, no RX interrupt if BF bit is reset
n (n = a + b)	a (a > 0)	b (b > 0)	— TX interrupt on identifiers matching the TX filters, — RX interrupt on identifiers matching the RX filters, — all other identifiers discarded (no interrupt)
b (b > 0)	0	b	— RX interrupt on identifiers matching the filters, — TX interrupt on all other identifiers if BF bit is set, no TX interrupt if BF bit is reset

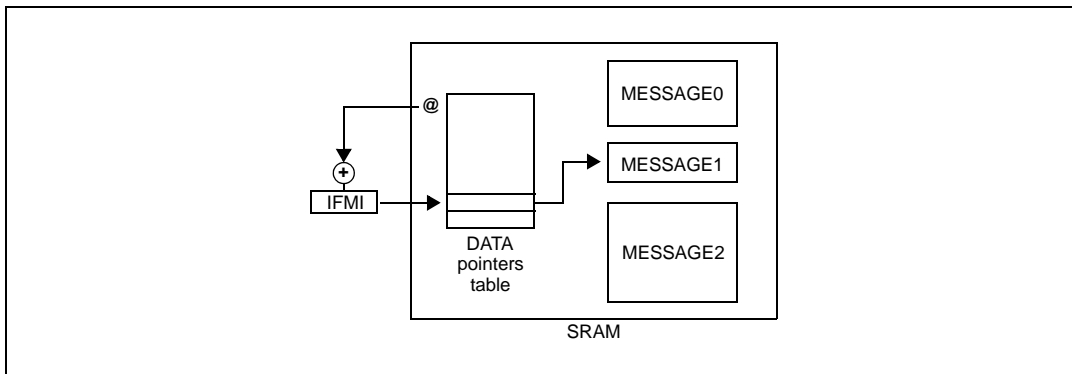


Figure 230. Identifier match index

**23.8.2.4 Slave mode with automatic resynchronization**

Automatic resynchronization must be enabled in Slave mode if  $f_{\text{periph\_set\_1\_clk}}$  tolerance is greater than 1.5%. This feature compensates a  $f_{\text{periph\_set\_1\_clk}}$  deviation up to 14%, as specified in LIN standard.

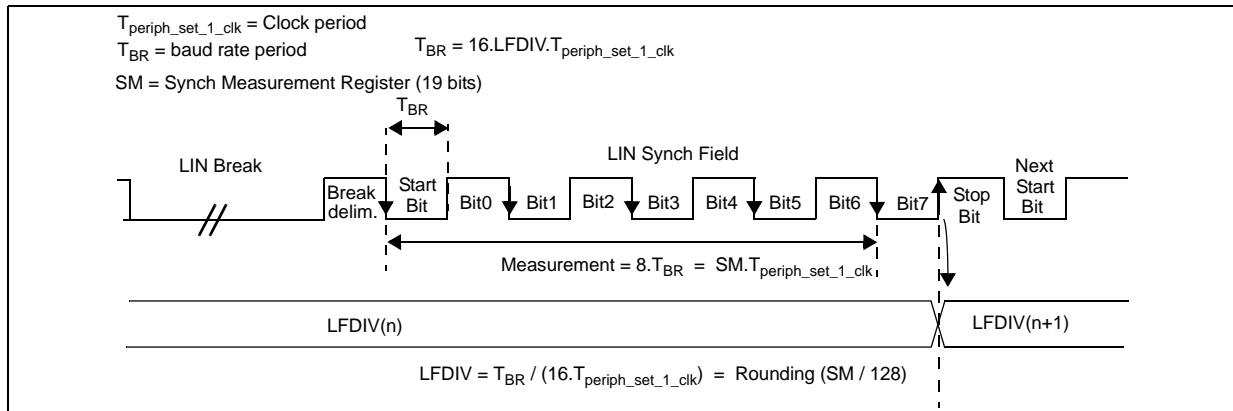
This mode is similar to Slave mode as described in [Section 23.8.2.2: Slave mode](#), with the addition of automatic resynchronization enabled by the LASE bit. In this mode LINFlex adjusts the fractional baud rate generator after each Synch Field reception.

Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN Break, the time duration between five falling edges on RDI is sampled on  $f_{\text{periph\_set\_1\_clk}}$  and the result of this measurement is stored in an internal 19-bit register called SM (not user accessible) (see [Figure 231](#)). Then the LFDIV value (and its associated registers LINIBRR and LINFBR) is



automatically updated at the end of the fifth falling edge. During LIN Synch Field measurement, the LINFlex state machine is stopped and no data is transferred to the data register.



**Figure 231. LIN synch field measurement**

LFDIV is an unsigned fixed point number. The mantissa is coded on 12 bits in the LINIBRR and the fraction is coded on 4 bits in the LINFBRR.

If LASE bit = 1 then LFDIV is automatically updated at the end of each LIN Synch Field.

Three internal registers (not user-accessible) manage the auto-update of the LINFlex divider (LFDIV):

- LFDIV\_NOM (nominal value written by software at LINIBRR and LINFBRR addresses)
- LFDIV\_MEAS (results of the Field Synch measurement)
- LFDIV (used to generate the local baud rate)

On transition to idle, break or break delimiter state due to any error or on reception of a complete frame, hardware reloads LFDIV with LFDIV\_NOM.

#### 23.8.2.4.1 Deviation error on the Synch Field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN Synch Field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the Synch Field:

- If  $D1 > 14.84\%$ , LHE is set.
- If  $D1 < 14.06\%$ , LHE is not set.
- If  $14.06\% < D1 < 14.84\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlex\_RX pin the  $f_{\text{periph\_set\_1\_clk}}$  clock.

The second check is based on a measurement of time between each falling edge of the Synch Field:

- If  $D2 > 18.75\%$ , LHE is set.
- If  $D2 < 15.62\%$ , LHE is not set.
- If  $15.62\% < D2 < 18.75\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlex\_RX pin the  $f_{\text{periph\_set\_1\_clk}}$  clock.

Note that the LINFlex does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full synch byte.

### 23.8.2.5 Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC\_ME). In LIN mode, the LINFlex controller acknowledges a clock gating request once the frame transmission or reception is completed.

## 23.8.3 8-bit timeout counter

### 23.8.3.1 LIN timeout mode

Resetting the LTOM bit in the LINTCSR enables the LIN timeout mode. The LINOCR becomes read-only, and OC1 and OC2 output compare values in the LINOCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the LINCR1[MME] bit), the 8-bit timeout counter will behave differently.

LIN timeout mode must not be enabled during LIN extended frames transmission or reception (that is, if the data field length in the BIDR is configured with a value higher than 8 data bytes).

#### 23.8.3.1.1 LIN Master mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO = 28-bit time.

Field OC1 checks  $T_{Header}$  and  $T_{Response}$  and field OC2 checks  $T_{Frame}$  (see [Figure 232](#)).

When LINFlex moves from Break delimiter state to Synch Field state (see [Section 23.7.1.3: LIN status register \(LINSR\)](#)):

- OC1 is updated with the value of  $OC_{Header}$  ( $OC_{Header} = CNT + 28$ ),
- OC2 is updated with the value of  $OC_{Frame}$  ( $OC_{Frame} = CNT + 28 + RTO \times 9$  (frame timeout value for an 8-byte frame)),
- the TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of  $OC_{Response}$  ( $OC_{Response} = CNT + RTO \times 9$  (response timeout value for an 8-byte frame)).

On the first response byte is received, OC1 and OC2 are automatically updated to check  $T_{Response}$  and  $T_{Frame}$  according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or response, the TOCE bit is reset.

If there is no response, frame timeout value does not take into account the DFL value, and an 8-byte response (DFL = 7) is always assumed.

**23.8.3.1.2 LIN Slave mode**

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO.

OC1 checks  $T_{Header}$  and  $T_{Response}$  and OC2 checks  $T_{Frame}$  (see [Figure 232](#)).

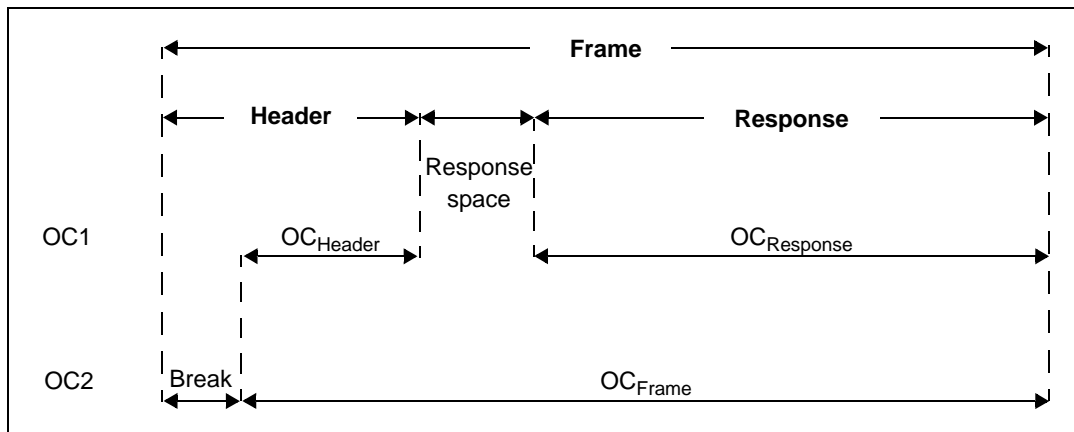
When LINFlex moves from Break state to Break Delimiter state (see [Section 23.7.1.3: LIN status register \(LINSR\)](#)):

- OC1 is updated with the value of  $OC_{Header}$  ( $OC_{Header} = CNT + HTO$ ),
- OC2 is updated with the value of  $OC_{Frame}$  ( $OC_{Frame} = CNT + HTO + RTO \times 9$  (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of  $OC_{Response}$  ( $OC_{Response} = CNT + RTO \times 9$  (response timeout value for an 8-byte frame)).

Once the first response byte is received, OC1 and OC2 are automatically updated to check  $T_{Response}$  and  $T_{Frame}$  according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or data field, the TOCE bit is reset.



**Figure 232. Header and response timeout**

**23.8.3.2 Output compare mode**

Programming LINTCSR[LTOM] = 1 enables the output compare mode. This mode allows the user to fully customize the use of the counter.

OC1 and OC2 output compare values can be updated in the LINTOCR by software.

## 23.8.4 Interrupts

**Table 233. LINFlex interrupt control**

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Header Received interrupt	HRF	HRIE	RXI <sup>(1)</sup>
Data Transmitted interrupt	DTF	DTIE	TXI
Data Received interrupt	DRF	DRIE	RXI
Data Buffer Empty interrupt	DBEF	DBEIE	TXI
Data Buffer Full interrupt	DBFF	DBFIE	RXI
Wake-up interrupt	WUPF	WUPIE	RXI
LIN State interrupt <sup>(2)</sup>	LSF	LSIE	RXI
Buffer Overrun interrupt	BOF	BOIE	ERR
Framing Error interrupt	FEF	FEIE	ERR
Header Error interrupt	HEF	HEIE	ERR
Checksum Error interrupt	CEF	CEIE	ERR
Bit Error interrupt	BEF	BEIE	ERR
Output Compare interrupt	OCF	OCIE	ERR
Stuck at Zero interrupt	SZF	SZIE	ERR

1. In Slave mode, if at least one filter is configured as TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.
2. For debug and validation purposes.

## 24 LIN Controller (LINFlexD)

### 24.1 Introduction

The LINFlexD (Local Interconnect Network Flexible with DMA support) controller interfaces the LIN network and supports the LIN protocol versions 1.3, 2.0, 2.1 and J2602 in both Master and Slave modes. LINFlexD includes a LIN mode that provides additional features (compared to standard UART) to ease LIN implementation, improve system robustness, minimize CPU load and allow slave node resynchronization.

Figure 233 shows the LINFlexD block diagram.

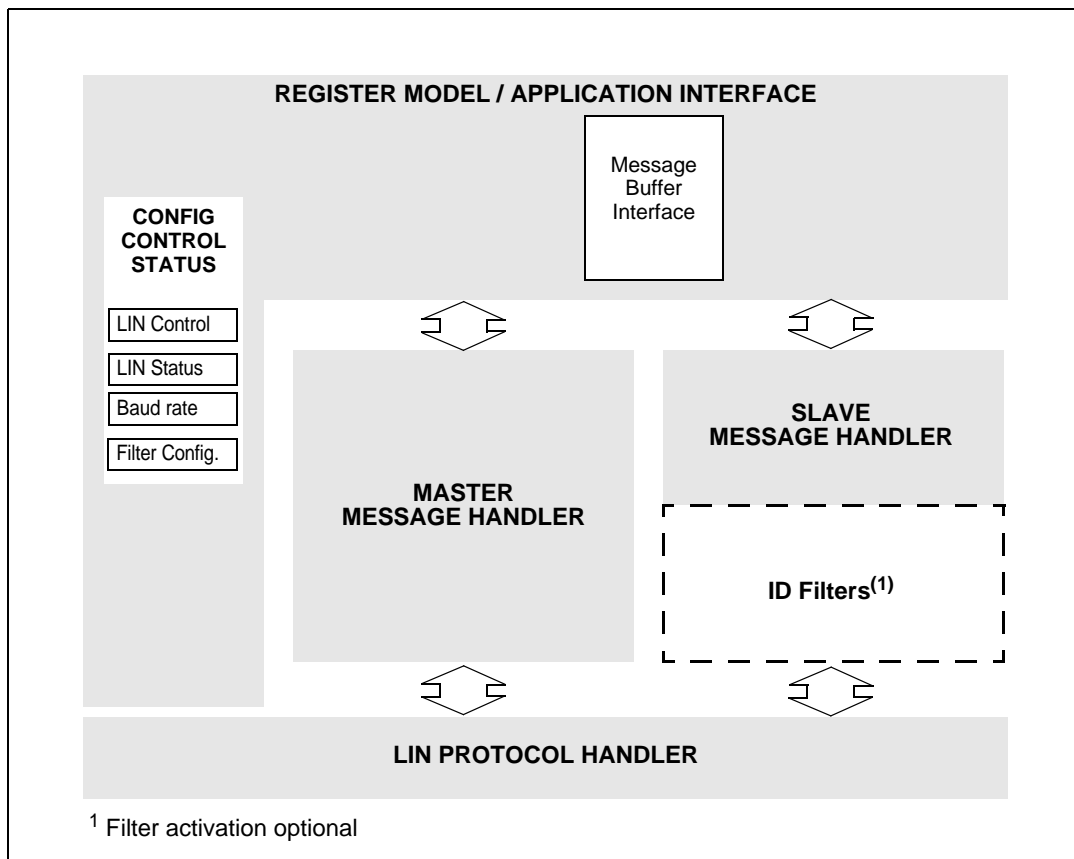


Figure 233. LINFlexD block diagram

### 24.2 Main features

The LINFlexD controller can operate in several modes, each of which has a distinct set of features. These distinct features are described in the following sections.

In addition, the LINFlexD controller has several features common to all modes:

- Fractional baud rate generator
- 3 operating modes for power saving and configuration registers lock
  - Initialization
  - Normal
  - Sleep
- 2 test modes
  - Loop Back
  - Self Test
- Maskable interrupts

### 24.2.1 LIN mode features

- Supports LIN protocol versions 1.3, 2.0, 2.1 and J2602
- Master mode with autonomous message handling
- Classic and enhanced checksum calculation and check
- Single 8-byte buffer for transmission/reception
- Extended frame mode for In-application Programming purposes
- Wake-up event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with FIRC as clock source
- Identifier filters for autonomous message handling in Slave mode

### 24.2.2 UART mode features

- Full-duplex communication
- Selectable frame size:
  - 8-bit frame
  - 9-bit frame
  - 16-bit frame
  - 17-bit frame
- Selectable parity:
  - Even
  - Odd
  - 0
  - 1
- 4-byte buffer for reception, 4-byte buffer for transmission
- 12-bit counter for timeout management

## 24.3 The LIN protocol

The LIN (Local Interconnect Network) is a serial communication protocol. The topology of a LIN network is shown in [Figure 234](#). A LIN network consists of:

- One master node
- Several slave nodes
- The LIN bus

A master node contains the master task as well as a slave task, all other nodes contain a slave task only. The master node decides when and which frame shall be transferred on the bus. The slave task provides the data to be transported by the frame.

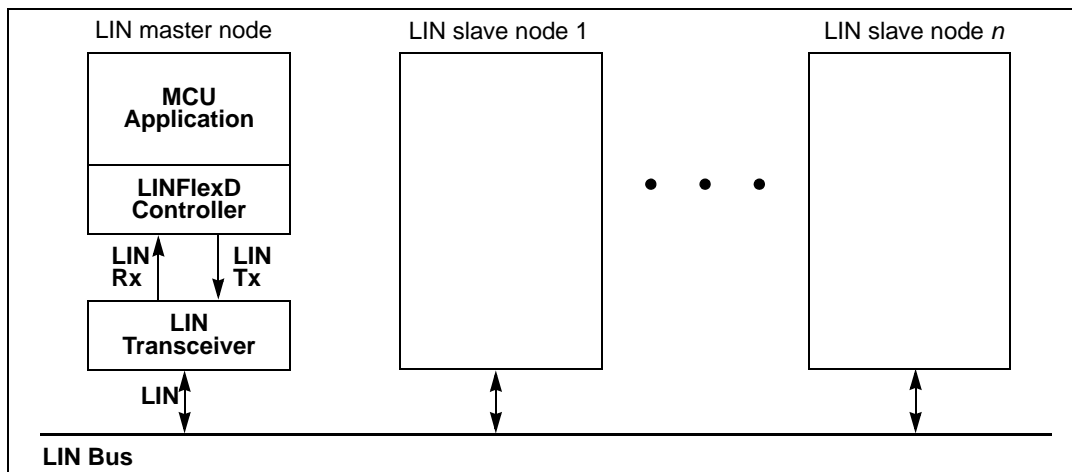


Figure 234. LIN network topology

### 24.3.1 Dominant and recessive logic levels

The LIN bus defines two logic levels, “dominant” and “recessive”, as follows:

- Dominant: logical low level (0)
- Recessive: logical high level (1)

### 24.3.2 LIN frames

A frame consists of a header provided by the master task and a response provided by the slave task, as shown in [Figure 235](#).

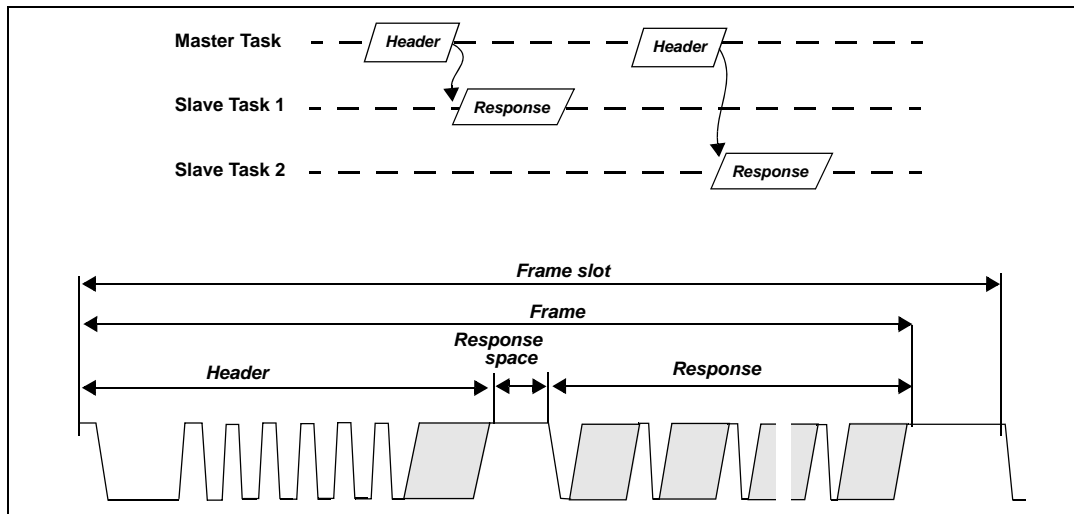


Figure 235. LIN frame structure

### 24.3.3 LIN header

The header consists of:

- A break field (described in [Section 24.3.3.1: Break field](#))
- A sync byte field (described in [Section 24.3.3.2: Sync byte field](#))
- An identifier (described in [Section 24.3.4.2: Identifier](#))

The slave task associated with the identifier provides the response.

#### 24.3.3.1 Break field

The break field, shown in [Figure 236](#), is used to signal the beginning of a new frame. It is always generated by the master and consists of:

- At least 13 dominant bits including the start bit
- At least one recessive bit that functions as break delimiter

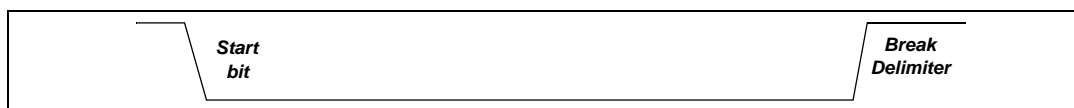


Figure 236. Break field

#### 24.3.3.2 Sync byte field

The sync pattern is a byte consisting of alternating dominant and recessive bits as shown in [Figure 237](#). It forms a data value of 0x55.

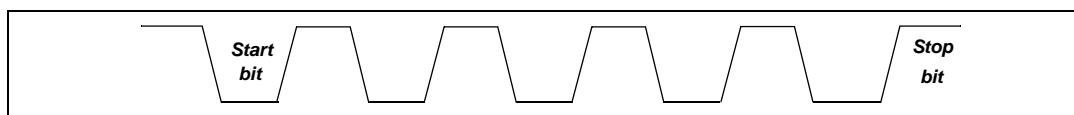


Figure 237. Sync pattern



### 24.3.4 Response

The response consists of:

- A data field (described in [Section 24.3.4.1: Data field](#))
- A checksum (described in [Section 24.3.4.3: Checksum](#))

The slave task interested in the data associated with the identifier receives the response and verifies the checksum.

#### 24.3.4.1 Data field

The structure of the data field transmitted on the LIN bus is shown in [Figure 238](#). The LSB of the data is sent first and the MSB last. The start bit is encoded as a dominant bit and the stop bit is encoded as a recessive bit.

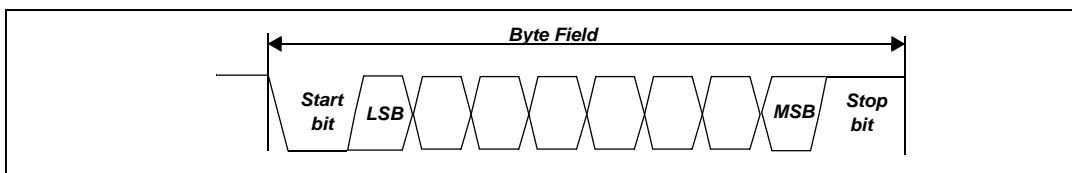


Figure 238. Structure of the data field

#### 24.3.4.2 Identifier

The identifier, shown in [Figure 239](#), consists of two sub-fields:

- The identifier value (in bits 0–5)
- The identifier parity (in bits 6–7)

The parity bits P0 and P1 are defined as follows:

- $P0 = ID0 \text{ xor } ID1 \text{ xor } ID2 \text{ xor } ID4$
- $P1 = \text{not}(ID1 \text{ xor } ID3 \text{ xor } ID4 \text{ xor } ID5)$

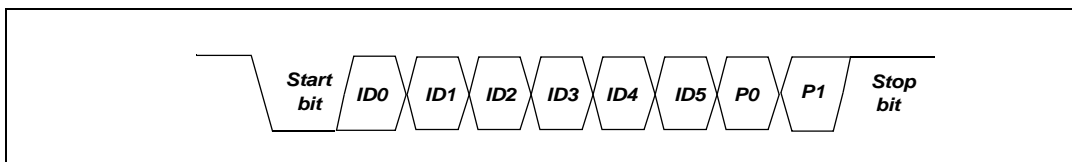


Figure 239. Identifier

#### 24.3.4.3 Checksum

The checksum contains the inverted 8-bit sum (with carry) over one of two possible groups:

- The classic checksum sums all data bytes, and is used for communication with LIN 1.3 slaves.
- The enhanced checksum sums all data bytes and the identifier, and is used for communication with LIN 2.0 (or later) slaves.

## 24.4 LINFlexD and software intervention

The increasing number of communication peripherals embedded on microcontrollers (for example, CAN, LIN, SPI) requires more and more CPU resources for the communication management. Even a 32-bit microcontroller is overloaded if its peripherals do not provide high level features to autonomously handle the communication.

Even though the LIN protocol with a maximum baud rate of 20 Kbit/s is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as is usually the case.

To minimize the CPU load in Master mode, LINFlexD handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlexD does not request any software (that is, application) intervention until the next header transmission request in transmission mode or until the checksum reception in reception mode.

To minimize the CPU load in Slave mode, LINFlexD requires software intervention only to:

- Trigger transmission or reception or data discard depending on the identifier
- Write data into the buffer (transmission mode) or read data from the buffer (reception mode) after checksum reception

If filter mode is activated for Slave mode, LINFlexD requires software intervention only to write data into the buffer (transmission mode) or read data from the buffer (reception mode)

The software uses the control, status and configuration registers to:

- Configure LIN parameters (for example, baud rate or mode)
- Request transmissions
- Handle receptions
- Manage interrupts
- Configure LIN error and timeout detection
- Process diagnostic information

The message buffer stores transmitted or received LIN frames.

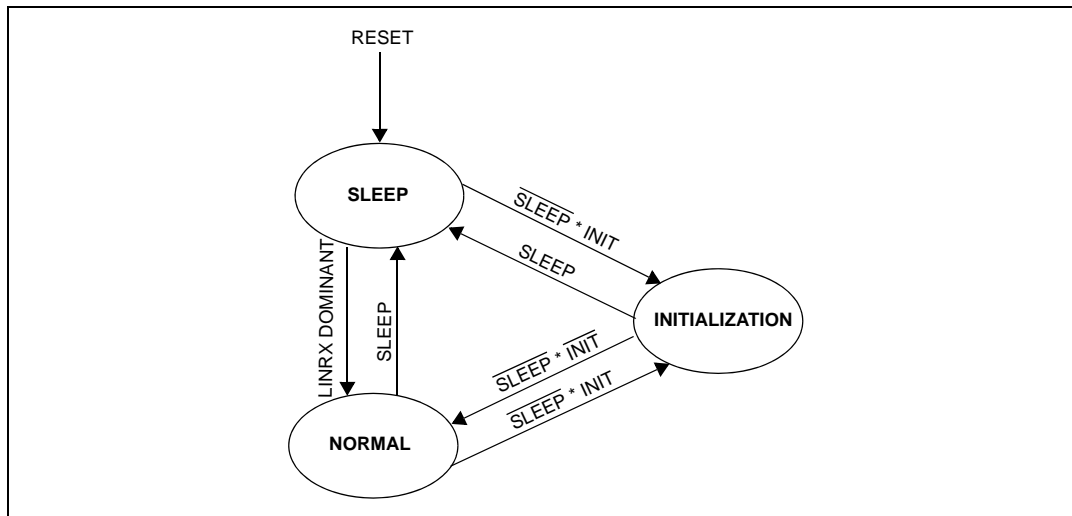
## 24.5 Summary of operating modes

The LINFlexD controller has three operating modes:

- Normal
- Initialization
- Sleep

After a hardware reset, the LINFlexD controller is in Sleep mode to reduce power consumption.

The transitions between these modes are shown in [Figure 240](#). The software instructs LINFlexD to enter Initialization mode or Sleep mode by setting LINCR1[INIT] or LINCR1[SLEEP], respectively.



**Figure 240. LINFlexD controller operating modes**

In addition to these controller-level operating modes, the LINFlexD controller also supports several protocol-level modes:

- LIN mode:
  - Master mode
  - Slave mode
  - Slave mode with identifier filtering
  - Slave mode with automatic resynchronization
- UART mode
- Test modes:
  - Loop Back mode
  - Self Test mode

These modes are discussed in detail in subsequent sections.

## 24.6 Controller-level operating modes

### 24.6.1 Initialization mode

The software initialization can be done while the hardware is in Initialization mode. To enter or exit this mode, the software sets or clears LINCR1[INIT], respectively.

In Initialization mode, all message transfers to and from the LIN bus are stopped and the LIN bus output (LINTX) is recessive.

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlexD controller, the software must:

- Select the desired mode (Master, Slave or UART)
- Set up the baud rate register
- If LIN Slave mode with filter activation is selected, initialize the identifier list

## 24.6.2 Normal mode

After initialization is complete, the software must clear LINCR1[INIT] to put the LINFlexD controller into Normal mode.

## 24.6.3 Sleep (low-power) mode

To reduce power consumption, LINFlexD has a low-power mode called Sleep mode. In this mode, the LINFlexD clock is stopped. Consequently, the LINFlexD will not update the status bits, but software can still access the LINFlexD registers.

To enter this mode, the software must set LINCR1[SLEEP].

LINFlexD can be awakened (exit Sleep mode) in one of two ways:

- The software clears LINCR1[SLEEP]
- Automatic wake-up is enabled (LINCR1[AWUM] is set) and LINFlexD detects LIN bus activity (that is, if a wake-up pulse of 150  $\mu$ s is detected on the LIN bus)

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing LINCR1[SLEEP] if LINCR1[AWUM] is set. To exit from Sleep mode if LINCR1[AWUM] is cleared, the software must clear LINCR1[SLEEP] when a wake-up event occurs.

## 24.7 LIN modes

### 24.7.1 Master mode

In Master mode, the software uses the message buffer to handle the LIN messages.

Master mode is selected when LINCR1[MME] is set.

#### 24.7.1.1 LIN header transmission

According to the LIN protocol, any communication on the LIN bus is triggered by the master sending a header. The header is transmitted by the master task while the data is transmitted by the slave task of a node.

To transmit a header with LINFlexD the application must set up the identifier, the data field length and configure the message (direction and checksum type) in the BIDR register before requesting the header transmission by setting LINCR2[HTRQ].

#### 24.7.1.2 Data transmission (transceiver as publisher)

When the master node is publisher of the data corresponding to the identifier sent in the header, then the slave task of the master has to send the data in the Response part of the LIN frame. Therefore, the software must provide the data to LINFlexD before requesting the header transmission. The software stores the data in the message buffer BDR. According to the data field length LINFlexD transmits the data and the checksum. The software uses the BIDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

If the response has been sent successfully, LINSR[DTF] is set. In case of error, the DTF flag is not set and the corresponding error flag is set in the LINESR (refer to Error handling). It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data

bytes, LINSR[DBEF] is set once the first 8 bytes have been transmitted. The application has to update the buffer BDR before resetting the DBEF bit. The transmission of the next bytes starts when the DBEF bit is reset. Once the last data byte (or the checksum byte) has been sent, the DTF flag is set.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the software sets this bit the response is sent by LINFlexD (publisher). Clearing this bit configures the message buffer as subscriber.

**24.7.1.3 Data reception (transceiver as subscriber)**

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlexD stores the data received from the slave in the message buffer and stores the message status in the LINSR. If the response has been received successfully, the LINSR(DRF) bit is set. In case of error, the DRF flag is not set and the corresponding error flag is set in the LINESR (refer to Error handling). It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR(DBFF) bit is set once the first 8 bytes have been received. The application has to read the buffer BDR before resetting the DBFF bit. Once the last data byte (or the checksum byte) has been received, the DRF flag is set.

**24.7.1.4 Data discard**

To discard data from a slave, the DIR bit in the BIDR must be reset and the DDRQ bit in LINC2 must be set before starting the header transmission.

**24.7.1.5 Error detection and handling**

LINFlexD is able to detect and handle LIN communication errors. A code stored in the LIN error status register (LINESR) signals the errors to the software.

*Table 234* lists the errors detected in Master mode and the LINFlexD controller’s response to these errors.

**Table 234. Errors in Master mode**

Error	Description	LINFlexD response to error
Bit error	During transmission, the value read back from the bus differs from the transmitted value	<ul style="list-style-type: none"> <li>– Stops the transmission of the frame after the corrupted bit</li> <li>– Generates an interrupt if LINIER[BEIE] is set</li> <li>– Returns to idle state</li> </ul>
Framing error	A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field)	If encountered during reception: <ul style="list-style-type: none"> <li>– Discards the current frame</li> <li>– Generates an interrupt if LINIER[FEIE] is set</li> <li>– Returns immediately to idle state</li> </ul>

Table 234. Errors in Master mode (continued)

Error	Description	LINFlexD response to error
Checksum error	The computed checksum does not match the received checksum	If encountered during reception: <ul style="list-style-type: none"> <li>– Discards the current frame</li> <li>– Generates an interrupt if LINIER[CEIE] is set</li> <li>– Returns to idle state</li> </ul>
Response and frame timeout	Refer to <a href="#">Section 24.12.1: 8-bit timeout counter</a> , for more details	

### 24.7.1.6 Overrun

Once the message buffer is full, the next valid message reception leads to an overrun and a message is lost. The hardware sets the BOF bit in the LINSR to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINCR1[RBLM] = 0) the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled (LINCR1[RBLM] = 1) the most recent message is discarded and the previous message is available in the buffer.

### 24.7.2 Slave mode

In Slave mode the software uses the message buffer to handle the LIN messages.

Slave mode is selected when the LINCR1[MME] is cleared.

#### 24.7.2.1 Data transmission (transceiver as publisher)

When LINFlexD receives the identifier, an RX interrupt is generated. The software must:

- Read the received ID in the BIDR register
- Fill the BDR registers
- Specify the data field length using the BIDR[DFL] field
- Trigger the data transmission by setting LINCR2[DTRQ]

One or several identifier filters can be configured for transmission by setting the DIR bits in the corresponding IFCR registers and activated by setting one or several bits in the IFER register.

When at least one identifier filter is configured in transmission and activated, and if the received ID matches the filter, a specific TX interrupt is generated.

Typically, the software has to copy the data from RAM locations to the BDRL and BDRM registers. To copy the data to the right location, the software has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter which matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer which points to the right data array in the RAM area and copy this data to the BDRL and BDRM registers (see [Figure 242](#)).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BDIR register. The software fills the BDRL and BDRM registers and triggers the data transmission by setting LINCR2[DTRQ].

If LINFlexD cannot provide enough TX identifier filters to handle all identifiers the software has to transmit data for, then a filter can be configured in mask mode (refer to [Section 24.7.3: Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

### 24.7.2.2 Data reception (transceiver as subscriber)

When LINFlexD receives the identifier, an RX interrupt is generated. The software must:

- Read the received ID in the BDIR register
- Specify the data field length using the BDIR[DFL] field before the reception of the stop bit of the first byte of data field

When the checksum reception is completed, an RX interrupt is generated to allow the software to read the received data in the BDR registers.

One or several identifier filters can be configured for reception by clearing the DIR bit in the corresponding IFCR registers and activated by clearing one or several bits in the IFER register.

When at least one identifier filter is configured in reception and activated, and if the received ID matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the software has to copy the data from the BDRL and BDRM registers to RAM locations. To copy the data to the right location, the software has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter which matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer which points to the right data array in the RAM area and copy this data from the BDRL and BDRM registers to the RAM (see [Figure 242](#)).

Using a filter avoids the software reading the ID value in the BDIR register, and configuring the direction, the data field length and the checksum type in the BDIR register.

If LINFlexD cannot provide enough RX identifier filters to handle all identifiers the software has to receive the data for, then a filter can be configured in mask mode (refer to [Section 24.7.3: Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

### 24.7.2.3 Data discard

When LINFlexD receives the identifier, the LINSR(HRF) bit is set and, if the LINIER(HRIE) bit is set, an RX interrupt is generated. If the received identifier does not concern the node, the software must set LINCR2[DDRQ]. LINFlexD returns to idle state after bit DDRQ is set.

### 24.7.2.4 Error detection and handling

[Table 235](#) lists the errors detected in Slave mode and the LINFlexD controller's response to these errors.

**Table 235. Errors in Slave mode**

Error	Description	LINFlexD response to error
Bit error	During transmission, the value read back from the bus differs from the transmitted value	– Stops the transmission of the frame after the corrupted bit – Generates an interrupt if LINIER[BEIE] is set – Returns to idle state
Framing error	A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field)	If encountered during reception: – Discards the current frame – Generates an interrupt if LINIER[FEIE] is set – Returns immediately to idle state
Checksum error	The computed checksum does not match the received checksum	If encountered during reception: – Discards the received frame – Generates an interrupt if LINIER[CEIE] is set – Returns to idle state
Header error	An error occurred during header reception (break delimiter error, inconsistent sync field, header timeout)	If encountered during header reception, a break field error, an inconsistent sync field, or a timeout: – Discards the header – Generates an interrupt if LINIER[HEIE] is set – Returns to idle state

**24.7.2.5 Valid header**

A received header is considered as valid when it has been received correctly according to the LIN protocol.

If a valid break field and break delimiter come before the end of the current header, or at any time during a data field, the current header or data is discarded and the state machine synchronizes on this new break.

**24.7.2.6 Valid message**

A received or transmitted message is considered as valid when the data has been received or transmitted without error according to the LIN protocol.

**24.7.2.7 Overrun**

After the message buffer is full, the next valid message reception causes an overrun and a message is lost. The LINFlexD controller sets LINSR[BOF] to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINCR1[RBLM] cleared), the last message stored in the buffer is overwritten by the new incoming message. In this case, the latest message is always available to the software.
- If the buffer lock function is enabled (LINCR1[RBLM] set), the most recent message is discarded and the previous message is available in the buffer.



### 24.7.3 Slave mode with identifier filtering

In the LIN protocol, the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. When a slave node receives a header, it decides - depending on the identifier value - whether the software needs to receive or send a response. If the message does not target the node, it must be discarded without software intervention.

To fulfill this requirement, the LINFlexD controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources which would otherwise be needed by software for filtering.

The filtering is accomplished through the use of IFCR registers. These registers have the names IFCR0 through IFCR15. This section also uses the nomenclature IFCR<sub>2n</sub> and IFCR<sub>2n+1</sub>; in this nomenclature, n is an integer, and the corresponding IFCR register is calculated using the formula in the subscript.

#### 24.7.3.1 Filter submodes

Usually each of the 16 IFCRs is used to filter one dedicated identifier, but this means that the LINFlexD controller could filter a maximum of 16 identifiers. In order to be able to handle more identifiers, the filters can be configured to operate as masks.

[Table 236](#) describes the two available filter submodes.

**Table 236. Filter submodes**

Submode	Description
Identifier list	Both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register. This is the default submode for the LINFlexD controller.
Mask	The identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”.

The bit mapping and register organization in these two submodes is shown in [Figure 241](#).

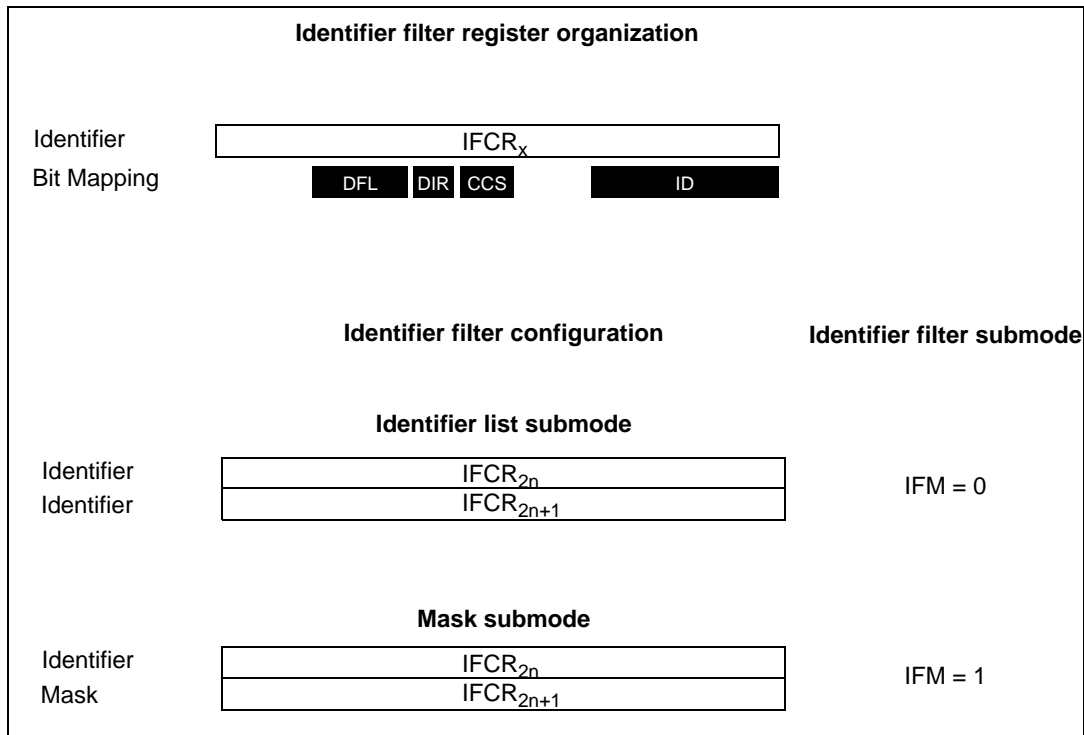


Figure 241. Filter configuration - register organization

**24.7.3.2 Identifier filter submode configuration**

The identifier filters are configured in the IFCR registers. To configure an identifier filter the filter must first be activated by setting the corresponding bit in the IFER[FACT] field. The submode (identifier list or mask) for the corresponding IFCR register is configured by the IFMR[IFM] field. For each filter, the IFCR register is used to configure:

- The ID or mask
- The direction (TX or RX)
- The data field length
- The checksum type

If no filter is active, an RX interrupt is generated on any received identifier event.

If at least one active filter is configured as TX, all received identifiers matching this filter generate a TX interrupt.

If at least one active filter is configured as RX, all received identifiers matching this filter generate an RX interrupt.

If no active filter is configured as RX, all received identifiers not matching TX filter(s) generate an RX interrupt.

Further details are provided in [Table 237](#) and [Figure 242](#).

Table 237. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as TX	Number of active filters configured as RX	Interrupt vector
0	0	0	- RX interrupt on all IDs
a (a > 0)	a	0	- TX interrupt on IDs matching the filters, - RX interrupt on all other IDs if BF bit is set, no RX interrupt if BF bit is reset
n (n = a + b)	a (a > 0)	b (b > 0)	- TX interrupt on IDs matching the TX filters, - RX interrupt on IDs matching the RX filters, - all other IDs discarded (no interrupt)
b (b > 0)	0	b	- RX interrupt on IDs matching the filters, - TX interrupt on all other IDs if BF bit is set, no TX interrupt if BF bit is reset

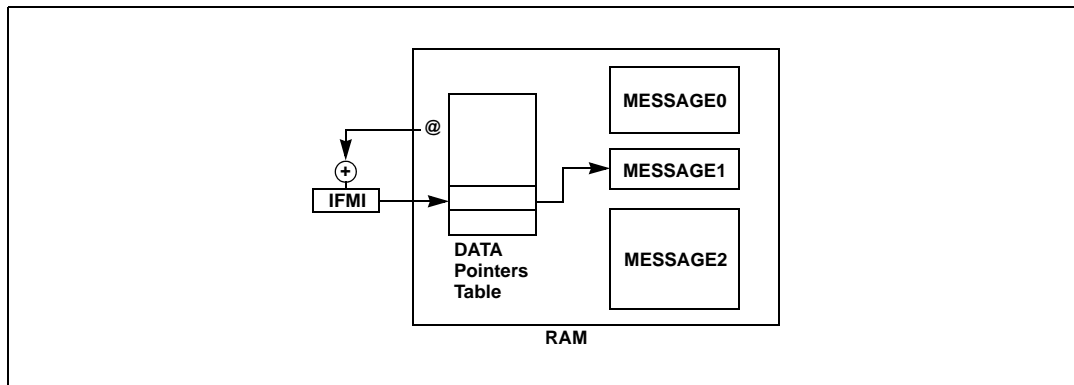


Figure 242. Identifier match index

### 24.7.4 Slave mode with automatic resynchronization

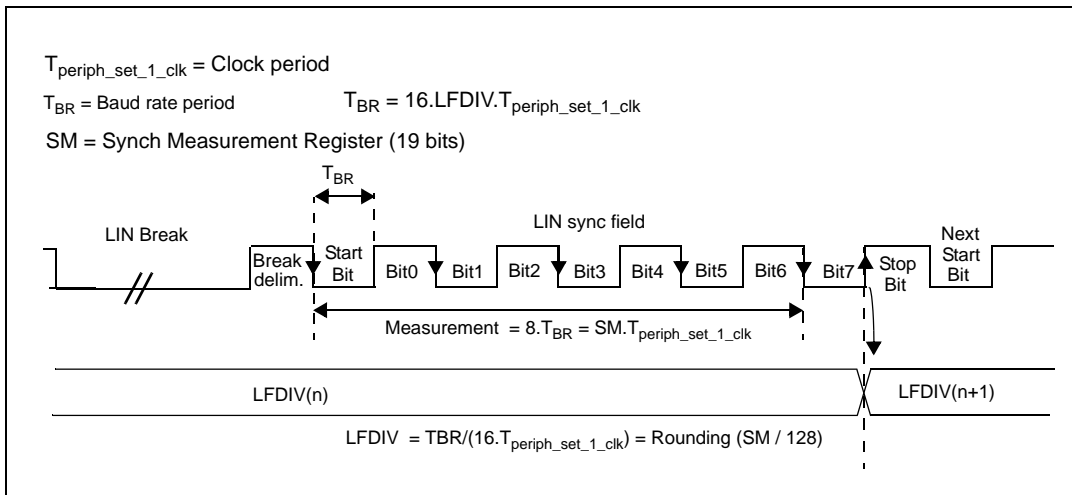
Automatic resynchronization must be enabled in Slave mode if  $f_{\text{ipg\_clock\_lin}}$  tolerance is greater than 1.5%. This feature compensates a  $f_{\text{periph\_set\_1\_clk}}$  deviation up to 14%, as specified in the LIN standard.

This mode is similar to Slave mode as described in [Section 24.7.2: Slave mode](#), with the addition of automatic resynchronization enabled by the LINCR1[LASE] bit. In this mode LINFlexD adjusts the fractional baud rate generator after each synch field reception.

#### 24.7.4.1 Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN break, the time duration between five falling edges on RDI is sampled on  $f_{\text{periph\_set\_1\_clk}}$  as shown in [Figure 243](#). Then the LFDIV value (and its associated LINIBRR and LINFBR registers) is automatically

updated at the end of the fifth falling edge. During LIN sync field measurement, the LINFlexD state machine is stopped and no data is transferred to the data register.



**Figure 243. LIN sync field measurement**

LFDIV is an unsigned fixed point number. The mantissa is coded on 20 bits in the LINIBRR register and the fraction is coded on 4 bits in the LINFBR register.

If LINCR1[LASE] is set, LFDIV is automatically updated at the end of each LIN sync field.

Three registers are used internally to manage the auto-update of the LINFlexD divider (LFDIV):

- LFDIV\_NOM (nominal value written by software at LINIBRR and LINFBR addresses)
- LFDIV\_MEAS (results of the Field Synch measurement)
- LFDIV (used to generate the local baud rate)

On transition to idle, break or break delimiter state due to any error or on reception of a complete frame, hardware reloads LFDIV with LFDIV\_NOM.

**24.7.4.2 Deviation error on the sync field**

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN sync field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the sync field:

- If  $D1 > 14.84\%$ , LHE is set.
- If  $D1 < 14.06\%$ , LHE is not set.
- If  $14.06\% < D1 < 14.84\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlexD\_RX pin the  $f_{\text{ipg\_clock\_lin}}$  clock.

The second check is based on a measurement of time between each falling edge of the sync field:

- If  $D2 > 18.75\%$ , LHE is set.
- If  $D2 < 15.62\%$ , LHE is not set.
- If  $15.62\% < D2 < 18.75\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlexD\_RX pin the  $f_{ipg\_clock\_lin}$  clock.

Note that the LINFlexD does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full synch byte.

### 24.7.4.3 Clock gating

The LINFlexD clock can be gated from the Mode Entry module (refer to Operating Modes chapter). In LIN mode, the LINFlexD controller acknowledges a clock gating request once the frame transmission or reception is completed.

## 24.8 Test modes

The LINFlexD controller includes two test modes, Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCR1 register. These bits must be configured while LINFlexD is in Initialization mode. After one of the two test modes has been selected, LINFlexD must be started in Normal mode.

### 24.8.1 Loop Back mode

LINFlexD can be put in Loop Back mode by setting LINCR1[LBKM]. In Loop Back mode, the LINFlexD treats its own transmitted messages as received messages. This is illustrated in [Figure 244](#).

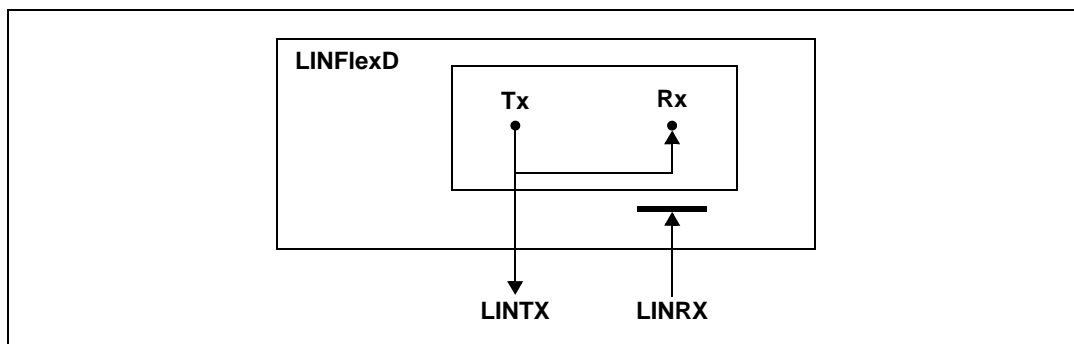


Figure 244. LINFlexD in Loop Back mode

This mode is provided for self-test functions. To be independent of external events, the LIN core ignores the LINRX signal. In this mode, the LINFlexD performs an internal feedback from its Tx output to its Rx input. The actual value of the LINRX input pin is disregarded by the LINFlexD. The transmitted messages can be monitored on the LINTX pin.

### 24.8.2 Self Test mode

LINFlexD can be put in Self Test mode by setting LINCR1[LBKM] and LINCR1[SFTM]. This mode can be used for a “Hot Self Test”, meaning the LINFlexD can be tested as in Loop

Back mode but without affecting a running LIN system connected to the LINTX and LINRX pins. In this mode, the LINRX pin is disconnected from the LINFlexD and the LINTX pin is held recessive. This is illustrated in [Figure 245](#).

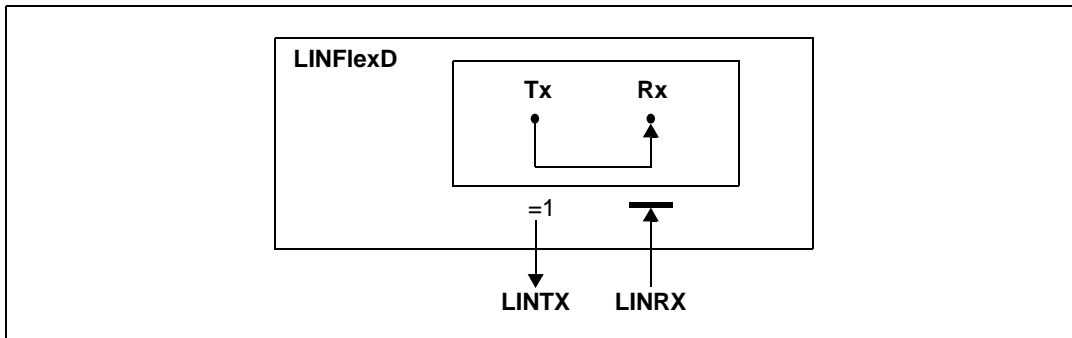


Figure 245. LINFlexD in Self Test mode

## 24.9 UART mode

The main features of UART mode are presented in [Section 24.2.2: UART mode features](#).

### 24.9.1 Data frame structure

#### 24.9.1.1 8-bit data frame

The 8-bit UART data frame is shown in [Figure 246](#). The 8th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. An even parity is set if the modulo-2 sum of the 7 data bits is 1. An odd parity is cleared in this case.

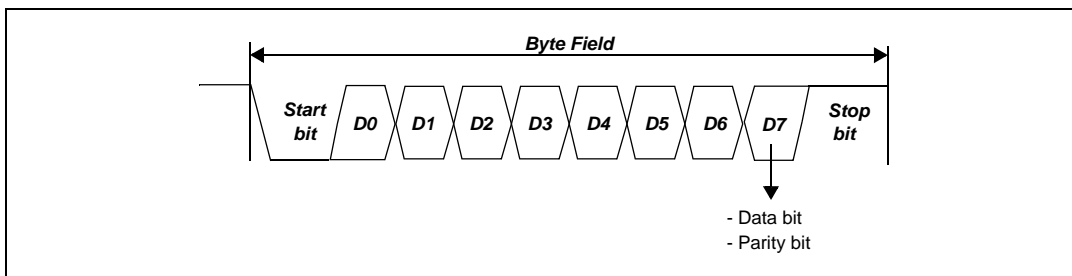


Figure 246. UART mode 8-bit data frame

#### 24.9.1.2 9-bit data frame

The 9-bit UART data frame is shown in [Figure 247](#). The 9th bit is a parity bit. Parity (even, odd, 0, or 1) can be selected by the by the UARTCR[PC] field. An even parity is set if the modulo-2 sum of the 8 data bits is 1. An odd parity is cleared in this case. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

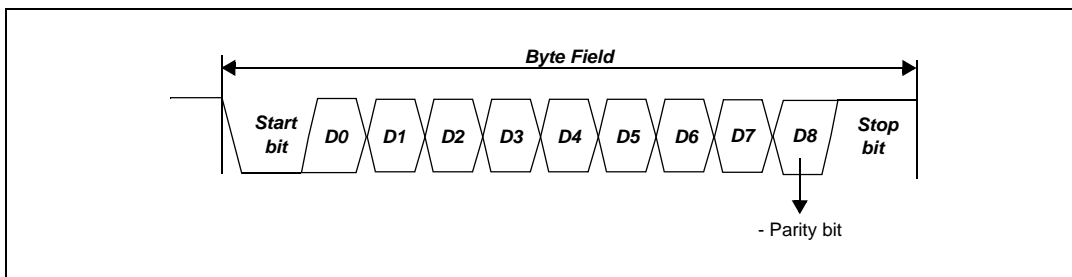


Figure 247. UART mode 9-bit data frame

24.9.1.3 16-bit data frame

The 16-bit UART data frame is shown in [Figure 248](#). The 16th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

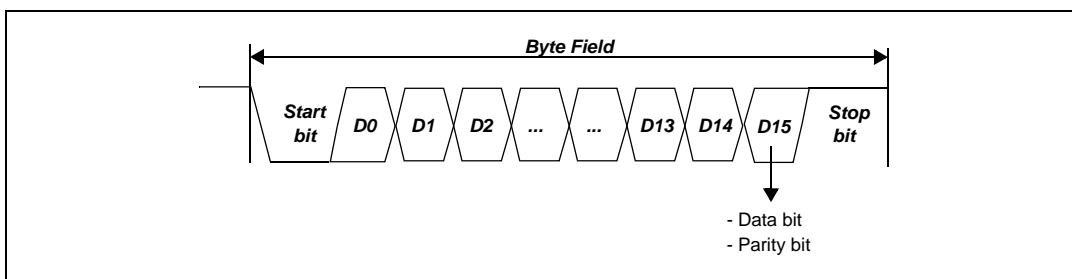


Figure 248. UART mode 16-bit data frame

24.9.1.4 17-bit data frame

The 17-bit UART data frame is shown in [Figure 249](#). The 17th bit is the parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

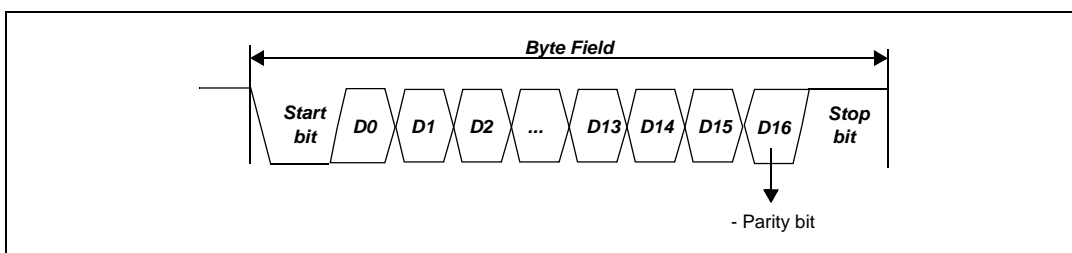


Figure 249. UART mode 17-bit data frame

24.9.2 Buffer

The 8-byte buffer is divided into two parts — one for receiver and one for transmitter — as shown in [Table 238](#).

**Table 238. UART buffer structure**

BDR	UART mode
0	Tx0
1	Tx1
2	Tx2
3	Tx3
4	Rx0
5	Rx1
6	Rx2
7	Rx3

For 16-bit frames, the lower 8 bits will be written in BDR0 and the upper 8 bits will be written in BDR1.

### 24.9.3 UART transmitter

In order to start transmission in UART mode, the UARTCR[UART] and UARTCR[TXEN] bits must be set. Transmission starts when BDR0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by the UARTCR[TDFLTC] field (see [Table 251](#)).

The Transmit buffer size is as follows:

- 4 bytes when UARTCR[WL1] = 0
- 2 half-words when UARTCR[WL1] = 1

Therefore, the maximum transmission that can be triggered is 4 bytes (2 half-words). After the programmed number of bytes has been transmitted, the UARTSR[DTFTFF] flag is set. If the UARTCR[TXEN] field is cleared during a transmission, the current transmission is completed, but no further transmission can be invoked. The buffer can be configured in FIFO mode (mandatory when DMA Tx is enabled) by setting UARTCR[TFBM].

The access to the BDRL register is shown in [Table 239](#).

**Table 239. BDRL access in UART mode**

Access	Mode <sup>(1)</sup>	Word length <sup>(2)</sup>	IPS operation result
Write Byte0	FIFO	Byte	OK
Write Byte1-2-3	FIFO	Byte	IPS transfer error
Write Half-word0-1	FIFO	Byte	IPS transfer error
Write Word	FIFO	Byte	IPS transfer error
Write Byte0-1-2-3	FIFO	Half-word	IPS transfer error
Write Half-word0	FIFO	Half-word	OK
Write Half-word1	FIFO	Half-word	IPS transfer error
Write Word	FIFO	Half-word	IPS transfer error
Read Byte0-1-2-3	FIFO	Byte/Half-word	IPS transfer error



**Table 239. BDRM access in UART mode (continued)**

Access	Mode <sup>(1)</sup>	Word length <sup>(2)</sup>	IPS operation result
Read Half-word0-1	FIFO	Byte/Half-word	IPS transfer error
Read Word	FIFO	Byte/Half-word	IPS transfer error
Write Byte0-1-2-3	BUFFER	Byte/Half-word	OK
Write Half-word0-1	BUFFER	Byte/Half-word	OK
Write Word	BUFFER	Byte/Half-word	OK
Read Byte0-1-2-3	BUFFER	Byte/Half-word	OK
Read Half-word0-1	BUFFER	Byte/Half-word	OK
Read Word	BUFFER	Byte/Half-word	OK

1. As specified by UARTCR[TFBM]
2. As specified by the WL1 and WL0 bits of the UARTCR register. In UART FIFO mode (UARTCR[TFBM] = 1), any read operation causes an IPS transfer error.

### 24.9.4 UART receiver

Reception of a data byte is started as soon as the software completes the following tasks in order:

1. Exits Initialization mode
2. Sets the UARTCR[RXEN] field
3. Detects the start bit

There is a dedicated data buffer for received data bytes. Its size is as follows:

- 4 bytes when UARTCR[WL1] = 0
- 2 half-words when UARTCR[WL1] = 1

After the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRFRFE] field is set. If the UARTCR[RXEN] field is cleared during a reception, the current reception is completed, but no further reception can be invoked until UARTCR[RXEN] is set again.

The buffer can be configured in FIFO mode (required when DMA Rx is enabled) by setting UARTCR[RFBM].

The access to the BDRM register is shown in [Table 240](#).

**Table 240. BDRM access in UART mode**

Access	Mode <sup>(1)</sup>	Word length <sup>(2)</sup>	IPS operation result
Read Byte4	FIFO	Byte	OK
Read Byte5-6-7	FIFO	Byte	IPS transfer error
Read Half-word2-3	FIFO	Byte	IPS transfer error
Read Word	FIFO	Byte	IPS transfer error
Read Byte4-5-6-7	FIFO	Half-word	IPS transfer error
Read Half-word2	FIFO	Half-word	OK
Read Half-word3	FIFO	Half-word	IPS transfer error

**Table 240. BDRM access in UART mode (continued)**

Access	Mode <sup>(1)</sup>	Word length <sup>(2)</sup>	IPS operation result
Read Word	FIFO	Half-word	IPS transfer error
Write Byte4-5-6-7	FIFO	Byte/Half-word	IPS transfer error
Write Half-word2-3	FIFO	Byte/Half-word	IPS transfer error
Write Word	FIFO	Byte/Half-word	IPS transfer error
Read Byte4-5-6-7	BUFFER	Byte/Half-word	OK
Read Half-word2-3	BUFFER	Byte/Half-word	OK
Read Word	BUFFER	Byte/Half-word	OK
Write Byte4-5-6-7	BUFFER	Byte/Half-word	IPS transfer error
Write Half-word2-3	BUFFER	Byte/Half-word	IPS transfer error
Write Word	BUFFER	Byte/Half-word	IPS transfer error

1. As specified by UARTCR[RFBM]
2. As specified by the WL1 and WL0 bits of the UARTCR register

*Table 241* lists some common scenarios, controller responses, and suggestions when the LINFlexD controller is acting as a UART receiver.

**Table 241. UART receiver scenarios**

Scenario	Responses and suggestions
The software does not know (in advance) how many bytes will be received.	Do not program UARTCR[RDFLRFC] in advance. When this field is zero (as it is after reset), reception occurs on a byte-by-byte basis. Therefore, the state machine will move to IDLE state after each byte is received.
UARTCR[RDFLRFC] is programmed for a certain number of bytes received, but the actual number of bytes received is smaller.	The reception will hang. In this case, the software must monitor the UARTSR[TO] field, and move to IDLE state by setting LINCR1[SLEEP].
A STOP request arrives before the reception is completed.	The request is acknowledged only after the programmed number of data bytes are received. In other words, the STOP request is not serviced immediately. In this case, the software must monitor the UARTSR[TO] field and move the state machine to IDLE state as appropriate. The stop request will be serviced only after this is complete.
A parity error occurs during the reception of a byte.	The corresponding UARTSR[PE <sub>n</sub> ] field is set. No interrupt is generated.

**Table 241. UART receiver scenarios (continued)**

Scenario	Responses and suggestions
A framing error occurs during the reception of a byte.	<ul style="list-style-type: none"> <li>– UARTSR[FE] is set.</li> <li>– If LINIER[FEIE] = 1, an interrupt is generated. This interrupt is helpful in identifying which byte has the framing error, since there is only one register bit for framing errors.</li> </ul>
A new byte has been received, but the last received frame has not been read from the buffer (UARTSR[RMB] has not yet been cleared by the software)	<ul style="list-style-type: none"> <li>– An overrun error will occur (UARTSR[BOF] will be set).</li> <li>– One message will be lost (depending on the setting of LINCR[RBLM]).</li> <li>– An interrupt is generated if LINIER[BOIE] is set.</li> </ul>

## 24.10 Memory map and register description

The memory maps for the LINFlexD modules on this microcontroller differ by module:

- The memory map for LINFlexD\_0 is shown in [Table 242](#).
- The memory map for LINFlexD\_1 is shown in [Table 243](#).

See the microcontroller memory map for the base addresses.

**Table 242. LINFlexD\_0 memory map**

Address offset	Register description	Location
0x00	LIN control register 1 (LINCR1)	<a href="#">on page 533</a>
0x04	LIN interrupt enable register (LINIER)	<a href="#">on page 536</a>
0x08	LIN status register (LINSR)	<a href="#">on page 538</a>
0x0C	LIN error status register (LINESR)	<a href="#">on page 541</a>
0x10	UART mode control register (UARTCR)	<a href="#">on page 542</a>
0x14	UART mode status register (UARTSR)	<a href="#">on page 545</a>
0x18	LIN timeout control status register (LINTCSR)	<a href="#">on page 547</a>
0x1C	LIN output compare register (LINOOCR)	<a href="#">on page 548</a>
0x20	LIN timeout control register (LINTOCR)	<a href="#">on page 549</a>
0x24	LIN fractional baud rate register (LINFBR)	<a href="#">on page 550</a>
0x28	LIN integer baud rate register (LINIBRR)	<a href="#">on page 550</a>
0x2C	LIN checksum field register (LINCFCR)	<a href="#">on page 551</a>
0x30	LIN control register 2 (LINCR2)	<a href="#">on page 552</a>
0x34	Buffer identifier register (BIDR)	<a href="#">on page 553</a>
0x38	Buffer data register least significant (BDRL)	<a href="#">on page 554</a>
0x3C	Buffer data register most significant (BDRM)	<a href="#">on page 555</a>
0x40	Identifier filter enable register (IFER)	<a href="#">on page 556</a>
0x44	Identifier filter match index (IFMI)	<a href="#">on page 557</a>

**Table 242. LINFlexD\_0 memory map (continued)**

Address offset	Register description	Location
0x48	Identifier filter mode register (IFMR)	<a href="#">on page 558</a>
0x4C–0x88	Identifier filter control registers 0–15 (IFCR0–IFCR15)	<a href="#">on page 559</a>
0x8C	Global control register (GCR)	<a href="#">on page 560</a>
0x90	UART preset timeout register (UARTPTO)	<a href="#">on page 561</a>
0x94	UART current timeout register (UARTCTO)	<a href="#">on page 562</a>
0x98	DMA Tx enable register (DMATXE)	<a href="#">on page 563</a>
0x9C	DMA Rx enable register (DMARXE)	<a href="#">on page 563</a>

**Table 243. LINFlexD\_1 memory map**

Address offset	Register description	Location
0x00	LIN control register 1 (LINC1R)	<a href="#">on page 533</a>
0x04	LIN interrupt enable register (LINIER)	<a href="#">on page 536</a>
0x08	LIN status register (LINSR)	<a href="#">on page 538</a>
0x0C	LIN error status register (LINESR)	<a href="#">on page 541</a>
0x10	UART mode control register (UARTCR)	<a href="#">on page 542</a>
0x14	UART mode status register (UARTSR)	<a href="#">on page 545</a>
0x18	LIN timeout control status register (LINTCSR)	<a href="#">on page 547</a>
0x1C	LIN output compare register (LINOOCR)	<a href="#">on page 548</a>
0x20	LIN timeout control register (LINTOCR)	<a href="#">on page 549</a>
0x24	LIN fractional baud rate register (LINFBR)	<a href="#">on page 550</a>
0x28	LIN integer baud rate register (LINIBRR)	<a href="#">on page 550</a>
0x2C	LIN checksum field register (LINCFR)	<a href="#">on page 551</a>
0x30	LIN control register 2 (LINC2R)	<a href="#">on page 552</a>
0x34	Buffer identifier register (BIDR)	<a href="#">on page 553</a>
0x38	Buffer data register least significant (BDRL)	<a href="#">on page 554</a>
0x3C	Buffer data register most significant (BDRM)	<a href="#">on page 555</a>
0x40	Identifier filter enable register (IFER)	<a href="#">on page 556</a>
0x44	Identifier filter match index (IFMI)	<a href="#">on page 557</a>
0x48	Identifier filter mode register (IFMR)	<a href="#">on page 558</a>
0x4C	Global control register (GCR)	<a href="#">on page 560</a>
0x50	UART preset timeout register (UARTPTO)	<a href="#">on page 561</a>
0x54	UART current timeout register (UARTCTO)	<a href="#">on page 562</a>
0x58	DMA Tx enable register (DMATXE)	<a href="#">on page 563</a>
0x5C	DMA Rx enable register (DMARXE)	<a href="#">on page 563</a>

### 24.10.1 LIN control register 1 (LINC1)

Offset:0x00 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CCD <sup>(1)</sup>	CFD <sup>(1)</sup>	LASE <sup>(1)</sup>	AWUM <sup>(1)</sup>	MBL <sup>(1)</sup>				BF <sup>(1)</sup>	SFTM <sup>(1)</sup>	LBKM <sup>(1)</sup>	MME <sup>(1)</sup>	SBDT <sup>(1)</sup>	RBLM <sup>(1)</sup>	SLEEP	INIT
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0/1 <sup>(2)</sup>	0	0	1	0

**Figure 250. LIN control register 1 (LINC1)**

1. These fields are writable only in Initialization mode (LINC1[INIT] = 1).
2. Resets to 0 in Slave mode and to 1 in Master mode.

**Table 244. LINC1 field descriptions**

Field	Description
CCD	Checksum Calculation disable This bit is used to disable the checksum calculation (see <a href="#">Table 245</a> ). 0: Checksum calculation is done by hardware. When this bit is reset the LINC1FR register is read-only. 1: Checksum calculation is disabled. When this bit is set the LINC1FR register is read/write. User can program this register to send a software calculated CRC (provided CFD is reset). Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
CFD	Checksum field disable This bit is used to disable the checksum field transmission (see <a href="#">Table 245</a> ). 0: Checksum field is sent after the required number of data bytes is sent. 1: No checksum field is sent. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
LASE	LIN Slave Automatic Resynchronization Enable 0: Automatic resynchronization disable 1: Automatic resynchronization enable Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
AWUM	Automatic Wake-Up Mode This bit controls the behavior of the LINFlexD hardware during Sleep mode. 0: The Sleep mode is exited on software request by clearing the SLEEP bit of the LINC1R register. 1: The Sleep mode is exited automatically by hardware on RX dominant state detection. The SLEEP bit of the LINC1R register is cleared by hardware whenever WUF bit in LINSR is set. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.

**Table 244. LINCR1 field descriptions (continued)**

Field	Description
MBL	LIN Master Break Length These bits indicate the Break length in Master mode (see <a href="#">Table 246</a> ). Note: These bits can be written in Initialization mode only. They are read-only in Normal or Sleep mode.
BF	Bypass filter 0: No interrupt if ID does not match any filter 1: An RX interrupt is generated on ID not matching any filter Notes: – If no filter is activated, this bit is reserved and always reads 1. – This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SFTM	Self Test Mode This bit controls the Self Test mode. For more details please refer to <a href="#">Section 24.8.2: Self Test mode</a> . 0: Self Test mode disable 1: Self Test mode enable Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
LBKM	Loop Back Mode This bit controls the Loop Back mode. For more details please refer to <a href="#">Section 24.8.1: Loop Back mode</a> . 0: Loop Back mode disable 1: Loop Back mode enable Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode
MME	Master Mode Enable 0: Slave mode enable 1: Master mode enable Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SBDT	Slave Mode Break Detection Threshold 0: 11-bit break 1: 10-bit break Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
RBLM	Receive Buffer Locked Mode 0: Receive Buffer not locked on overrun. Once the Slave Receive Buffer is full the next incoming message overwrites the previous one. 1: Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SLEEP	Sleep Mode Request This bit is set by software to request LINFlexD to enter Sleep mode. This bit is cleared by software to exit Sleep mode or by hardware if the AWUM bit in LINCR1 and the WUF bit in LINSR are set (see <a href="#">Table 247</a> ).
INIT	Initialization Request The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlexD enters Normal mode when clearing the INIT bit (see <a href="#">Table 247</a> ).

**Table 245. Checksum bits configuration**

CFD	CCD	LINCFR	Checksum sent
1	1	Read/Write	None
1	0	Read-only	None
0	1	Read/Write	Programmed in LINCFR by bits CF[0:7]
0	0	Read-only	Hardware calculated

**Table 246. LIN master break length selection**

MBL	Length
0000	10-bit
0001	11-bit
0010	12-bit
0011	13-bit
0100	14-bit
0101	15-bit
0110	16-bit
0111	17-bit
1000	18-bit
1001	19-bit
1010	20-bit
1011	21-bit
1100	22-bit
1101	23-bit
1110	36-bit
1111	50-bit

**Table 247. Operating mode selection**

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

### 24.10.2 LIN interrupt enable register (LINIER)

Offset: 0x04

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZIE	OCIE	BEIE	CEIE	HEIE	0	0	FEIE	BOIE	LSIE	WUIE	DBFIE	DBEITOE	DRIE	DTIE	HRIE
W	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 251. LIN interrupt enable register (LINIER)

Table 248. LINIER field descriptions

Field	Description
SZIE	Stuck at Zero Interrupt Enable 0: No interrupt when SZF bit in LINESR or UARTSR is set 1: Interrupt generated when SZF bit in LINESR or UARTSR is set
OCIE	Output Compare Interrupt Enable 0: No interrupt when OCF bit in LINESR or UARTSR is set 1: Interrupt generated when OCF bit in LINESR or UARTSR is set
BEIE	Bit Error Interrupt Enable 0: No interrupt when BEF bit in LINESR is set 1: Interrupt generated when BEF bit in LINESR is set
CEIE	Checksum Error Interrupt Enable 0: No interrupt on Checksum error 1: Interrupt generated when checksum error flag (CEF) is set in LINESR
HEIE	Header Error Interrupt Enable 0: No interrupt on Break Delimiter error, Synch Field error, ID field error 1: Interrupt generated on Break Delimiter error, Synch Field error, ID field error
FEIE	Framing Error Interrupt Enable 0: No interrupt on Framing error 1: Interrupt generated on Framing error
BOIE	Buffer Overrun Interrupt Enable 0: No interrupt on Buffer overrun 1: Interrupt generated on Buffer overrun



**Table 248. LINIER field descriptions (continued)**

Field	Description
LSIE	LIN State Interrupt Enable 0: No interrupt on LIN state change 1: Interrupt generated on LIN state change This interrupt can be used for debugging purposes. It has no status flag but is reset when writing '1111' into the LIN state bits in the LINSR register.
WUIE	Wake-up Interrupt Enable 0: No interrupt when WUF bit in LINSR or UARTSR is set 1: Interrupt generated when WUF bit in LINSR or UARTSR is set
DBFIE	Data Buffer Full Interrupt Enable 0: No interrupt when buffer data register is full 1: Interrupt generated when data buffer register is full
DBEIETOIE	Data Buffer Empty Interrupt Enable / Timeout Interrupt Enable 0: No interrupt when buffer data register is empty 1: Interrupt generated when data buffer register is empty <b>Note:</b> An interrupt is generated if this bit is set and one of the following is true: LINFlexD is in LIN mode and LINSR[DBEF] is set LINFlexD is in UART mode and UARTSR[TO] is set
DRIE	Data Reception Complete Interrupt Enable 0: No interrupt when data reception is completed 1: Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set
DTIE	Data Transmitted Interrupt Enable 0: No interrupt when data transmission is completed 1: Interrupt generated when data transmitted flag (DTF) is set in LINSR or UARTSR register
HRIE	Header Received Interrupt Enable 0: No interrupt when a valid LIN header has been received 1: Interrupt generated when a valid LIN header has been received, that is, HRF bit in LINSR register is set

### 24.10.3 LIN status register (LINSR)

Offset: 0x08

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LINS				0	0	RMB	0	RBSY	RPS	WUF	DBFF	DBEF	DRF	DTF	HRF
W	w1c						w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 252. LIN status register (LINSR)

**Table 249. LINSR field descriptions**

Field	Description
LINS	<p>LIN state</p> <p>LIN mode states description</p> <p>0000: Sleep mode LINFlexD is in Sleep mode to save power consumption.</p> <p>0001: Initialization mode LINFlexD is in Initialization mode.</p> <p>0010: Idle This state is entered on several events:</p> <ul style="list-style-type: none"> <li>– SLEEP bit and INIT in LINCR1 register have been cleared by software,</li> <li>– A falling edge has been received on RX pin and AWUM bit is set,</li> <li>– The previous frame reception or transmission has been completed or aborted.</li> </ul> <p>0011: Break In Slave mode, a falling edge followed by a dominant state has been detected. Receiving Break. Note: In Slave mode, in case of error new LIN state can be either Idle or Break depending on last bit state. If last bit is dominant new LIN state is Break, otherwise Idle. In Master mode, Break transmission ongoing.</p> <p>0100: Break Delimiter In Slave mode, a valid Break has been detected. Refer to LINCR1 register for break length configuration (10-bit or 11-bit). Waiting for a rising edge.</p> <p>In Master mode, Break transmission has been completed. Break Delimiter transmission is ongoing.</p> <p>0101: Synch Field In Slave mode, a valid Break Delimiter has been detected (recessive state for at least one bit time). Receiving Synch Field. In Master mode, Synch Field transmission is ongoing.</p> <p>0110: Identifier Field In Slave mode, a valid Synch Field has been received. Receiving ID Field. In Master mode, identifier transmission is ongoing.</p> <p>0111: Header reception/transmission completed In Slave mode, a valid header has been received and identifier field is available in the BIDR register. In Master mode, header transmission is completed.</p> <p>1000: Data reception/transmission Response reception/transmission is ongoing.</p> <p>1001: Checksum Data reception/transmission completed. Checksum reception/transmission ongoing.</p> <p>In UART mode, only the following states are flagged by the LIN state bits:</p> <ul style="list-style-type: none"> <li>– Init</li> <li>– Sleep</li> <li>– Idle</li> <li>– Data transmission/reception</li> </ul>
RMB	<p>Release Message Buffer</p> <p>0: Buffer is free</p> <p>1: Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.</p>

**Table 249. LINSR field descriptions (continued)**

Field	Description
RBSY	Receiver Busy Flag 0: Receiver is Idle 1: Reception ongoing Note: In Slave mode, after header reception, if DIR bit in BIDR is reset and reception starts then this bit is set. In this case, user cannot set DTRQ bit in LINCR2.
RPS	LIN receive pin state This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a falling edge on the LINRX pin when – slave is in Sleep mode, – master is in Sleep mode or idle state. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if WUIE bit in LINIER is set.
DBFF	Data Buffer Full Flag This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL > 7). This bit must be cleared by software. It is reset by hardware in Initialization mode.
DBEF	Data Buffer Empty Flag This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL > 7). This bit must be cleared by software, once buffer has been filled again, in order to start transmission. This bit is reset by hardware in Initialization mode.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. Note: This flag is not set in case of bit error or framing error.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. Note: This flag is not set in case of bit error if IOBE bit is reset.
HRF	Header Reception Flag This bit is set by hardware and indicates a valid header reception is completed. This bit must be cleared by software. This bit is reset by hardware in Initialization mode and at end of completed or aborted frame. Note: If filters are enabled, this bit is set only when identifier software filtering is required, that is to say: – all filters are inactive and BF bit in LINCR1 is set – no match in any filter and BF bit in LINCR1 is set – TX filter match

### 24.10.4 LIN error status register (LINESR)

Offset: 0x0C

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	BEF	CEF	SFEF	BDEF	IDPEF	FEF	BOF	0	0	0	0	0	0	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 253. LIN error status register (LINESR)

Table 250. LINESR field descriptions

Field	Description
SZF	Stuck at zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.
OCF	Output Compare Flag 0: No output compare event occurred 1: The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. If this bit is set and IOT bit in LINTCSR is set, LINFlexD moves to Idle state. If LTOM bit in LINTCSR register is set then OCF is reset by hardware in Initialization mode. If LTOM bit is reset, then OCF maintains its status whatever the mode is.
BEF	Bit Error Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a bit error. This error can occur during response field transmission (Slave and Master modes) or during header transmission (in Master mode). This bit is cleared by software.
CEF	Checksum error Flag This bit is set by hardware and indicates that the received checksum does not match the hardware calculated checksum. This bit is cleared by software. Note: This bit is never set if CCD or CFD bit in LINCR1 register is set.
SFEF	Synch Field Error Flag This bit is set by hardware and indicates that a Synch Field error occurred (inconsistent Synch Field).
BDEF	Break Delimiter Error Flag This bit is set by hardware and indicates that the received Break Delimiter is too short (less than one bit time).

**Table 250. LINESR field descriptions (continued)**

Field	Description
IDPEF	Identifier Parity Error Flag This bit is set by hardware and indicates that a Identifier Parity error occurred. Note: Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a framing error (invalid stop bit). This error can occur during reception of any data in the response field (Master or Slave mode) or during reception of Synch Field or Identifier Field in Slave mode.
BOF	Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites the buffer. It can be cleared by software.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

**24.10.5 UART mode control register (UARTCR)**

Offset: 0x10 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
R	TDFLFC <sup>(1)</sup>				RDFLFC <sup>(1)</sup>				RFBM	TFBM <sup>(2)</sup>	WL[1] <sup>(2)</sup>	PC1 <sup>(2)</sup>	RXEN	TXEN	PC0 <sup>(2)</sup>	PCE <sup>(2)</sup>	WL[0] <sup>(2)</sup>	UART <sup>(2)</sup>	
W																			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 254. UART mode control register (UARTCR)**

1. These fields are read/write in UART buffer mode and read-only in other modes.
2. These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

**Table 251. UARTCR field descriptions**

Field	Description
TDFLTC	<p>Transmitter data field length / Tx FIFO counter</p> <p>This field has one of two functions depending on the mode of operation as follows:</p> <ul style="list-style-type: none"> <li>– When LINFlexD is in UART buffer mode (TFBM = 0), TDFLTC defines the number of bytes to be transmitted. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit):                      0bX00: 1 byte                      0bX01: 2 bytes                      0bX10: 3 bytes                      0bX11: 4 bytes</li> <li>When the UART data length is configured as half-word (WL = 0b10 or 0b11), the only valid values for TDFLTC are 0b001 and 0b011.</li> <li>– When LINFlexD is in UART FIFO mode (TFBM = 1), TDFLTC contains the number of entries (bytes) of the Tx FIFO. The field is read-only in this configuration. The permissible values are as follows:                      0b000: Empty                      0b001: 1 byte                      0b010: 2 bytes                      0b011: 3 bytes                      0b100: 4 bytes                      All other values are reserved.</li> </ul> <p>This field is meaningful and can be programmed only when the UART bit is set.</p>
RDFLRFC	<p>Receiver data field length / Rx FIFO counter</p> <p>This field has one of two functions depending on the mode of operation as follows:</p> <ul style="list-style-type: none"> <li>– When LINFlexD is in UART buffer mode (RFBM = 0), RDFLRFC defines the number of bytes to be received. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit):                      0bX00: 1 byte                      0bX01: 2 bytes                      0bX10: 3 bytes                      0bX11: 4 bytes</li> <li>When the UART data length is configured as half-word (WL = 0b10 or 0b11), the only valid values for RDFLRFC are 0b001 and 0b011.</li> <li>– When LINFlexD is in UART FIFO mode (RFBM = 1), RDFLRFC contains the number of entries (bytes) of the Rx FIFO. The field is read-only in this configuration. The permissible values are as follows:                      0b000: Empty                      0b001: 1 byte                      0b010: 2 bytes                      0b011: 3 bytes                      0b100: 4 bytes                      All other values are reserved.</li> </ul> <p>This field is meaningful and can be programmed only when the UART bit is set.</p>
RFBM	<p>Rx FIFO/buffer mode</p> <p>0 Rx buffer mode enabled                      1 Rx FIFO mode enabled (mandatory in DMA Rx mode)</p> <p>This field can be programmed in initialization mode only when the UART bit is set.</p>

**Table 251. UARTCR field descriptions (continued)**

Field	Description
TFBM	Tx FIFO/buffer mode 0 Tx buffer mode enabled 1 Tx FIFO mode enabled (mandatory in DMA Tx mode)  This field can be programmed in initialization mode only when the UART bit is set.
RXEN	Receiver Enable 0: Receiver disabled 1: Receiver enabled  This field can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0: Transmitter disabled 1: Transmitter enabled  This field can be programmed only when the UART bit is set. Note: Transmission starts when this bit is set and when writing DATA0 in the BDRL register.
PC	Parity control 00 Parity sent is even 01 Parity sent is odd 10 A logical 0 is always transmitted/checked as parity bit 11 A logical 1 is always transmitted/checked as parity bit  This field can be programmed in initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0: Parity transmit/check disabled 1: Parity transmit/check enabled  This field can be programmed in Initialization mode only when the UART bit is set.
WL	Word length in UART mode 00 7 bits data + parity 01 8 bits data when PCE = 0 or 8 bits data + parity when PCE = 1 10 15 bits data + parity 11 16 bits data when PCE = 0 or 16 bits data + parity when PCE = 1  This field can be programmed in Initialization mode only when the UART bit is set.
UART	UART mode enable 0: LIN mode 1: UART mode  This field can be programmed in Initialization mode only.



### 24.10.6 UART mode status register (UARTSR)

Offset: 0x14

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	PE3	PE2	PE1	PE0	RMB	FEF	BOF	RPS	WUF	0	TO	DRFRFE	DTFTFF	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 255. UART mode status register (UARTSR)

Table 252. UARTSR field descriptions

Field	Description
SZF	Stuck at zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.
OCF	OCF Output Compare Flag 0: No output compare event occurred 1: The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCCR. An interrupt is generated if the OCIE bit in LINIER register is set.
PE3	Parity Error Flag Rx3 This bit indicates if there is a parity error in the corresponding received byte (Rx3). No interrupt is generated if this error occurs. 0: No parity error 1: Parity error
PE2	Parity Error Flag Rx2 This bit indicates if there is a parity error in the corresponding received byte (Rx2). No interrupt is generated if this error occurs. 0: No parity error 1: Parity error
PE1	Parity Error Flag Rx1 This bit indicates if there is a parity error in the corresponding received byte (Rx1). No interrupt is generated if this error occurs. 0: No parity error 1: Parity error

**Table 252. UARTSR field descriptions (continued)**

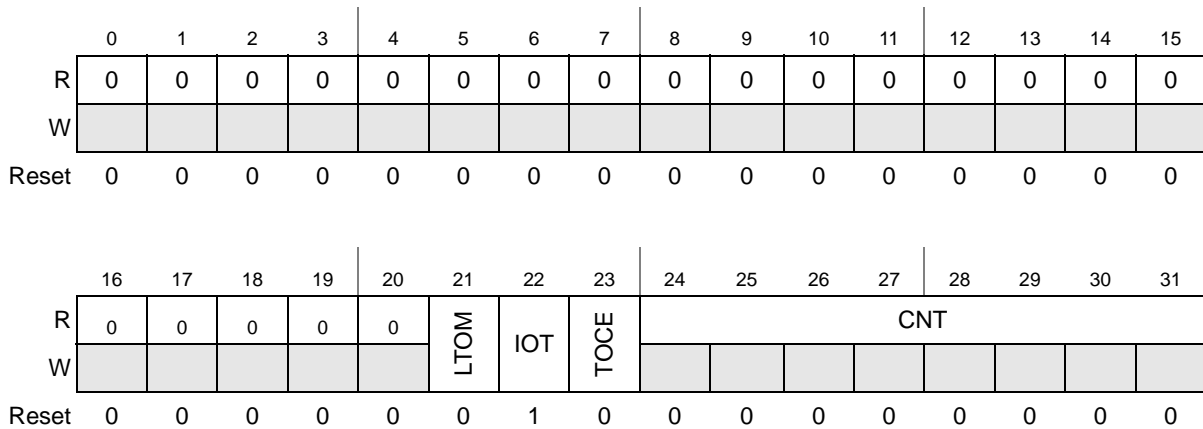
Field	Description
PE0	Parity Error Flag Rx0 This bit indicates if there is a parity error in the corresponding received byte (Rx0). No interrupt is generated if this error occurs. 0: No parity error 1: Parity error
RMB	Release Message Buffer 0: Buffer is free 1: Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a framing error (invalid stop bit).
BOF	FIFO/buffer overrun flag This bit is set by hardware when a new data byte is received and the RMB bit is not cleared in UART buffer mode. In UART FIFO mode, this bit is set when there is a new byte and the Rx FIFO is full. In UART FIFO mode, once Rx FIFO is full, the new received message is discarded regardless of the value of LINCR1[RBLM]. If LINCR1[RBLM] = 1, the new byte received is discarded. If LINCR1[RBLM] = 0, the new byte overwrites buffer. This field can be cleared by writing a 1 to it. An interrupt is generated if LINIER[BOIE] is set.
RPS	LIN Receive Pin State This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a falling edge on the LINRX pin in Sleep mode. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if WUIE bit in LINIER is set.
TO	Timeout The LINFlexD controller sets this field when a UART timeout occurs — that is, when the value of UARTCTO becomes equal to the preset value of the timeout (UARTPTO register setting). This field should be cleared by software. The GCR[SR] field should be used to reset the receiver FSM to idle state in case of UART timeout for UART reception depending on the application both in buffer and FIFO mode. An interrupt is generated when LINIER[DBEIETOIE] is set on the Error interrupt line in UART mode.
DRFRFE	Data reception completed flag / Rx FIFO empty flag The LINFlexD controller sets this field as follows: – In UART buffer mode (RFBM = 0), it indicates that the number of bytes programmed in RDFL has been received. This field should be cleared by software. An interrupt is generated if LINIER[DRIE] is set. This field is set in case of framing error, parity error, or overrun. This field reflects the same value as in LINESR when in Initialization mode and UART bit is set. – In UART FIFO mode (RFBM = 1), it indicates that the Rx FIFO is empty. This field is a read-only field used internally by the DMA Rx interface.

**Table 252. UARTSR field descriptions (continued)**

Field	Description
DTFTFF	Data transmission completed flag / Tx FIFO full flag The LINFlexD controller sets this field as follows: – In UART buffer mode (TFBM = 0), it indicates that the data transmission is completed. This field should be cleared by software. An interrupt is generated if LINIER[DTIE] is set. This field reflects the same value as in LINESR when in Initialization mode and UART bit is set. – In UART FIFO mode (TFBM = 1), it indicates that the Tx FIFO is full. This field is a read-only field used internally by the DMA Tx interface.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

**24.10.7 LIN timeout control status register (LINTCSR)**

Offset: 0x18 Access: User read/write



1. These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 256. LIN timeout control status register (LINTCSR)**

**Table 253. LINTCSR field descriptions**

Name	Description
LTOM	LIN timeout mode 0: LIN timeout mode (header, response and frame timeout detection) 1: Output compare mode This bit can be set/cleared in Initialization mode only.
IOT	Idle on Timeout 0: LIN state machine not reset to Idle on timeout event 1: LIN state machine reset to Idle on timeout event This bit can be set/cleared in Initialization mode only.

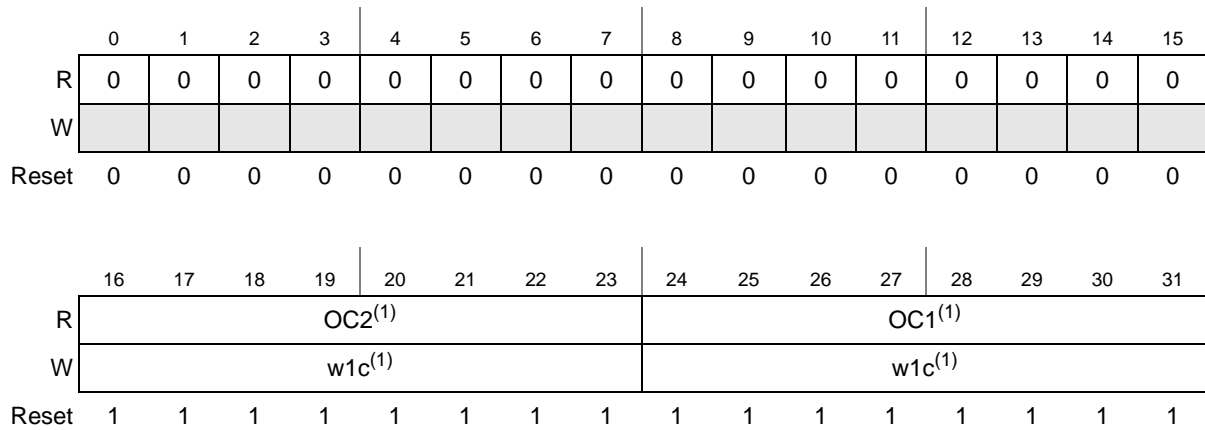
**Table 253. LINTCSR field descriptions (continued)**

Name	Description
TOCE	Timeout counter enable 0: Timeout counter disable. OCF bit in LINESR or UARTSR is not set on an output compare event. 1: Timeout counter enable. OCF bit is set if an output compare event occurs. TOCE bit is configurable by software in Initialization mode. If LIN state is not Init and if timer is in LIN timeout mode, then hardware takes control of TOCE bit.
CNT	Counter Value These bits indicate the LIN Timeout counter value.

**24.10.8 LIN output compare register (LINOCR)**

Offset: 0x1C

Access: User read/write



**Figure 257. LIN output compare register (LINOCR)**

1. If LINTCSR[LATOM] = 0, these fields are read-only.

**Table 254. LINOCR field descriptions**

Field	Description
OC2	Output compare 2 value These bits contain the value to be compared to the value of LINTCSR[CNT].
OC1	Output compare 1 value These bits contain the value to be compared to the value of LINTCSR[CNT].

### 24.10.9 LIN timeout control register (LINTOCR)

Offset: 0x20

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	0	0	0	0	RTO				0	HTO <sup>3</sup>							
W																	
Reset	0	0	0	0	1	1	1	0	0	0	0/1 <sup>1</sup>	0/1 <sup>2</sup>	1	1	0	0	

1. Resets to 1 in Slave mode and to 0 in Master mode
2. Resets to 0 in Slave mode and to 1 in Master mode
3. HTO field can only be written in slave mode, LINCR1[MME] = 0.

**Figure 258. LIN timeout control register (LINTOCR)**

**Table 255. LINTOCR field descriptions**

Field	Description
RTO	Response timeout value This register contains the response timeout duration (in bit time) for 1 byte. The reset value is 0xE = 14, corresponding to TResponse_Maximum = 1.4 x TResponse_Nominal
HTO	Header timeout value This register contains the header timeout duration (in bit time). This value does not include the first 11 dominant bits of the Break. The reset value depends on which mode LINFlexD is in. HTO can be written only for Slave mode.

### 24.10.10 LIN fractional baud rate register (LINFBR)

Offset: 0x24 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIV_F <sup>1</sup>			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This field is writable only in Initialization mode, LINCR1[INIT] = 1.

**Figure 259. LIN timeout control register (LINTOCR)**

**Table 256. LINFBR field descriptions**

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlexD divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F / 16.  This register can be written in Initialization mode only, LINCR1[INIT] = 1.

### 24.10.11 LIN integer baud rate register (LINIBRR)

Offset: 0x28 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIV_M <sup>1</sup>																			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This field is writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 260. LIN integer baud rate register (LINIBRR)**

**Table 257. LINIBRR field descriptions**

Field	Description
DIV_M	LFDIV mantissa These bits define the LINFlexD divider (LFDIV) mantissa value (see <a href="#">Table 258</a> ). This register can be written in Initialization mode only.

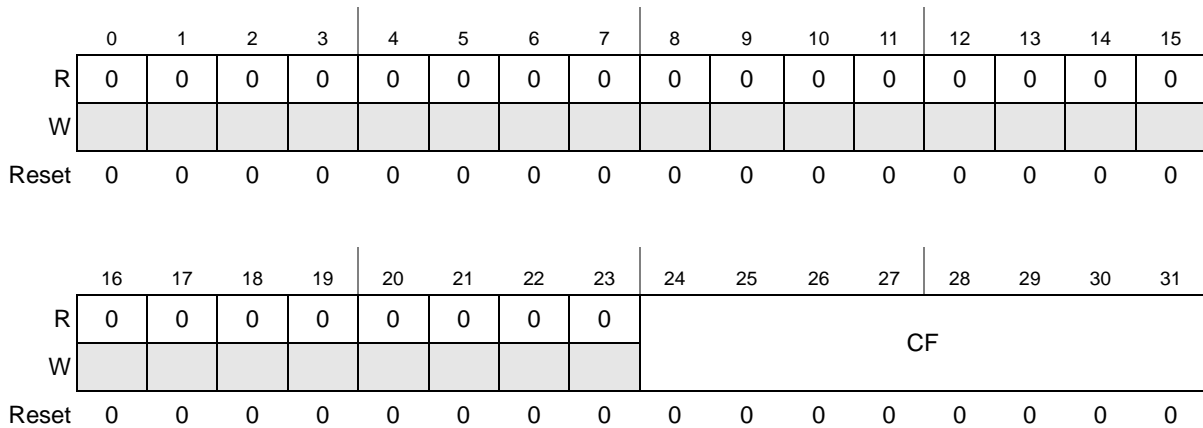
**Table 258. Integer baud rate selection**

DIV_M	Mantissa
0x0	LIN clock disabled
0x1	1
...	...
0xFFFFE	1048574
0xFFFFF	1048575

**24.10.12 LIN checksum field register (LINCFR)**

Offset: 0x2C

Access: User read/write



**Figure 261. LIN checksum field register (LINCFR)**

**Table 259. LINCFR field descriptions**

Field	Description
CF	Checksum bits When LINCR1[CCD] is cleared, these bits are read-only. When LINCR1[CCD] is set, these bits are read/write. See <a href="#">Table 245</a> .

24.10.13 LIN control register 2 (LINC2)

Offset: 0x30

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	IOBE <sup>1</sup>	IOPE <sup>1</sup>	WURQ	DDRQ	DTRQ	ABRQ	HTRQ	0	0	0	0	0	0	0	0
W				w1c	w1c	w1c	w1c	w1c								
Reset	0	1	0/1 <sup>2</sup>	0	0	0	0	0	0	0	0	0	0	0	0	0

1. These fields are writable only in Initialization mode (LINC2[INIT] = 1).
2. Resets to 1 in Slave mode and to 0 in Master mode

Figure 262. LIN control register 2 (LINC2)

Table 260. LINC2 field descriptions

Field	Description
IOBE	Idle on Bit Error 0: Bit error does not reset LIN state machine 1: Bit error reset LIN state machine This bit can be set/cleared in Initialization mode only (LINC2[INIT]) = 1.
IOPE	Idle on Identifier Parity Error 0: Identifier Parity error does not reset LIN state machine. 1: Identifier Parity error reset LIN state machine. This bit can be set/cleared in Initialization mode only (LINC2[INIT]) = 1.
WURQ	Wake-up Generation Request Setting this bit generates a wake-up pulse. It is reset by hardware when the wake-up character has been transmitted. The character sent is copied from DATA0 in BDRL buffer. Note that this bit cannot be set in Sleep mode. Software has to exit Sleep mode before requesting a wake-up. Bit error is not checked when transmitting the wake-up request.
DDRQ	Data Discard Request Set by software to stop data reception if the frame does not concern the node. This bit is reset by hardware once LINFlexD has moved to idle state. In Slave mode, this bit can be set only when HRF bit in LINSR is set and identifier did not match any filter.





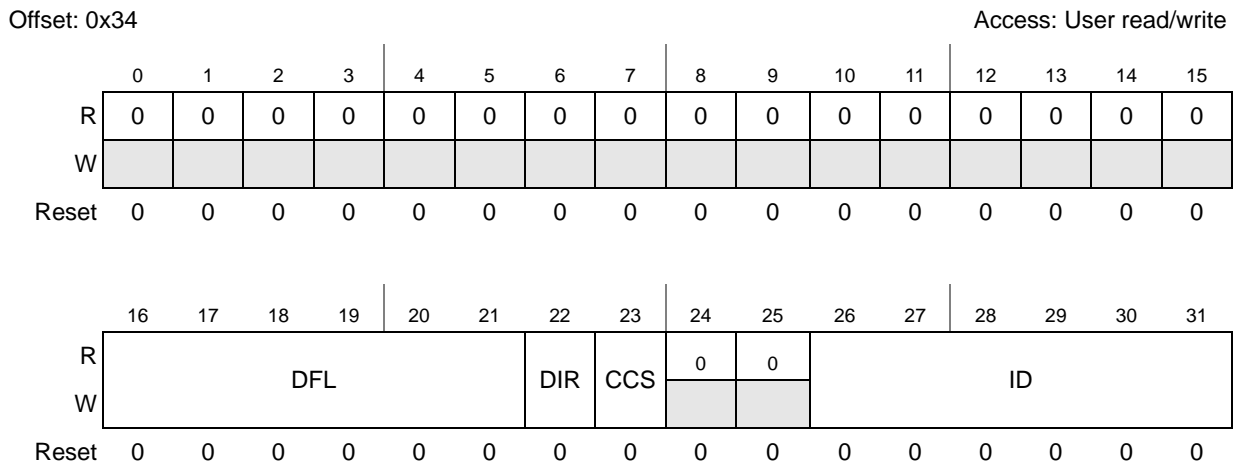
**Table 260. LINCR2 field descriptions (continued)**

Field	Description
DTRQ	Data Transmission Request Set by software in Slave mode to request the transmission of the LIN Data field stored in the Buffer data register. This bit can be set only when HRF bit in LINSR is set. Cleared by hardware when the request has been completed or aborted or on an error condition. In Master mode, this bit is set by hardware when DIR bit in BIDR is set and header transmission is completed.
ABRQ	Abort Request Set by software to abort the current transmission. Cleared by hardware when the transmission has been aborted. LINFlexD aborts the transmission at the end of the current bit. This bit can also abort a wake-up request. It can also be used in UART mode.
HTRQ	Header Transmission Request Set by software to request the transmission of the LIN header. Cleared by hardware when the request has been completed or aborted. This bit has no effect in UART mode.

**24.10.14 Buffer identifier register (BIDR)**

This register contains the fields that identify a transaction and provide other information related to it.

All the fields in this register must be updated when an ID filter (enabled) in slave mode (Tx or Rx) matches the ID received.

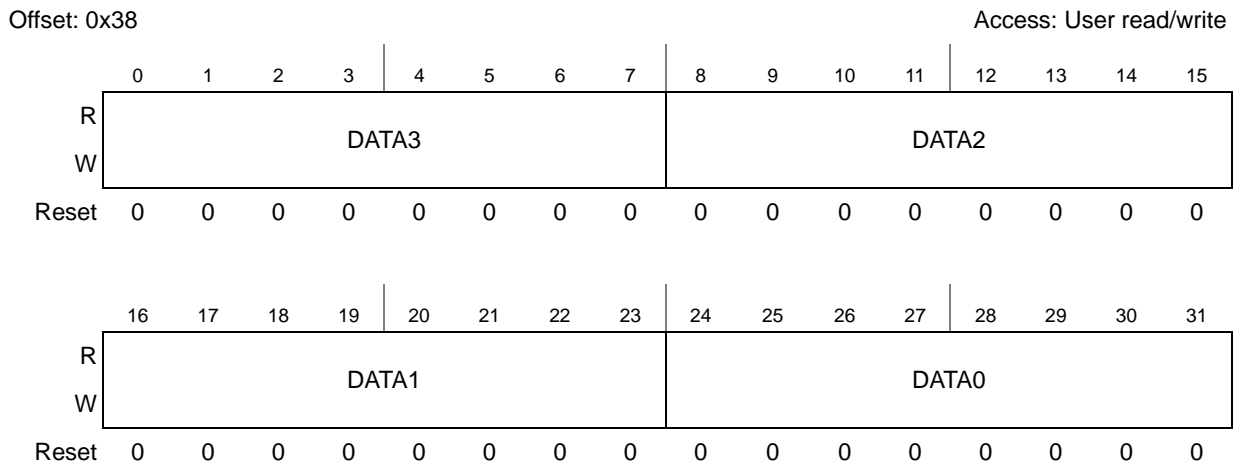


**Figure 263. Buffer identifier register (BIDR)**

**Table 261. BIDR field descriptions**

Field	Description
DFL	Data Field Length These bits define the number of data bytes in the response part of the frame. DFL = Number of data bytes - 1. Normally, LIN uses only DFL[0:2] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[0:2] and DFL[0:5]...DFL[3:5] are provided to manage extended frames.
DIR	Direction This bit controls the direction of the data field. 0: LINFlexD receives the data and copy them in the BDR registers. 1: LINFlexD transmits the data from the BDR registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0: Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1: Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and below.
ID	Identifier Identifier part of the identifier field without the identifier parity.

**24.10.15 Buffer data register least significant (BDRL)**



**Figure 264. Buffer data register least significant (BDRL)**

**Table 262. BDRL field descriptions**

Field	Description
DATA3	Data Byte 3 Data byte 3 of the data field
DATA2	Data Byte 2 Data byte 2 of the data field

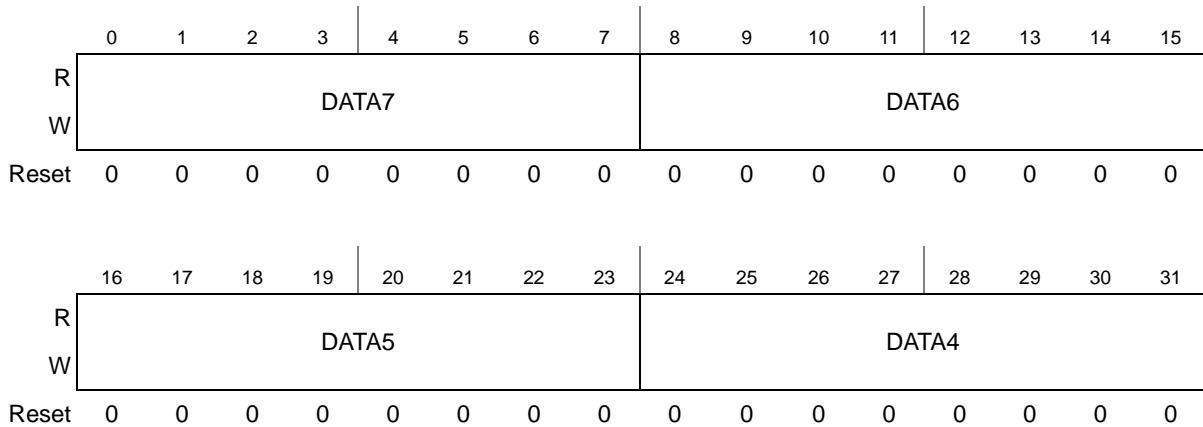
**Table 262. BDRL field descriptions (continued)**

Field	Description
DATA1	Data Byte 1 Data byte 1 of the data field
DATA0	Data Byte 0 Data byte 0 of the data field

**24.10.16 Buffer data register most significant (BDRM)**

Offset: 0x3C

Access: User read/write



**Figure 265. Buffer data register most significant (BDRM)**

**Table 263. BDRM field descriptions**

Field	Description
DATA7	Data Byte 7 Data byte 7 of the data field
DATA6	Data Byte 6 Data byte 6 of the data field
DATA5	Data Byte 5 Data byte 5 of the data field
DATA4	Data Byte 4 Data byte 4 of the data field

### 24.10.17 Identifier filter enable register (IFER)

Offset: 0x40 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	FACT <sup>(1)</sup>							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 266. Identifier filter enable register (IFER)**

1. This field is writable only in Initialization mode (LINC1[INIT] = 1).

**Table 264. IFER field descriptions**

Field	Description
FACT	Filter activation (see <a href="#">Table 265</a> ) The software sets the bit FACT[x] to activate the filters x in identifier list mode. In identifier mask mode bits FACT(2n + 1) have no effect on the corresponding filters as they act as masks for the Identifiers 2n. 0 Filters 2n and 2n + 1 are deactivated. 1 Filters 2n and 2n + 1 are activated.

**Table 265. IFER[FACT] configuration**

Bit	Value	Result
FACT[0]	0	Filters 0 and 1 are deactivated.
	1	Filters 0 and 1 are activated.
FACT[1]	0	Filters 2 and 3 are deactivated.
	1	Filters 2 and 3 are activated.
FACT[2]	0	Filters 4 and 5 are deactivated.
	1	Filters 4 and 5 are activated.
FACT[3]	0	Filters 6 and 7 are deactivated.
	1	Filters 6 and 7 are activated.
FACT[4]	0	Filters 8 and 9 are deactivated.
	1	Filters 8 and 9 are activated.

**Table 265. IFER[FACT] configuration (continued)**

Bit	Value	Result
FACT[5]	0	Filters 10 and 11 are deactivated.
	1	Filters 10 and 11 are activated.
FACT[6]	0	Filters 12 and 13 are deactivated.
	1	Filters 12 and 13 are activated.
FACT[7]	0	Filters 14 and 15 are deactivated.
	1	Filters 14 and 15 are activated.

**24.10.18 Identifier filter match index (IFMI)**

Offset: 0x44

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0		IFMI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 267. Identifier filter match index (IFMI)**

**Table 266. IFMI field descriptions**

Field	Description
IFMI	Filter match index This register contains the index corresponding to the received ID. It can be used to directly write or read the data in RAM (refer to <a href="#">Section 24.7.2: Slave mode</a> , for more details). When no filter matches, IFMI = 0. When Filter n is matching, IFMI = n + 1.

### 24.10.19 Identifier filter mode register (IFMR)

Offset:0x48 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	IFM							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 268. Identifier filter mode register (IFMR)**

**Table 267. IFMR field descriptions**

Field	Description
IFM	Filter mode 0 Filters $2n$ and $2n + 1$ are in identifier list mode. 1 Filters $2n$ and $2n + 1$ are in mask mode (filter $2n + 1$ is the mask for the filter $2n$ ). (See <a href="#">Table 268.</a> )

**Table 268. IFMR[IFM] configuration**

Bit	Value	Result
IFM[0]	0	Filters 0 and 1 are in identifier list mode.
	1	Filters 0 and 1 are in mask mode (filter 1 is the mask for the filter 0).
IFM[1]	0	Filters 2 and 3 are in identifier list mode.
	1	Filters 2 and 3 are in mask mode (filter 3 is the mask for the filter 2).
IFM[2]	0	Filters 4 and 5 are in identifier list mode.
	1	Filters 4 and 5 are in mask mode (filter 5 is the mask for the filter 4).
IFM[3]	0	Filters 6 and 7 are in identifier list mode.
	1	Filters 6 and 7 are in mask mode (filter 7 is the mask for the filter 6).
IFM[4]	0	Filters 8 and 9 are in identifier list mode.
	1	Filters 8 and 9 are in mask mode (filter 9 is the mask for the filter 8).
IFM[5]	0	Filters 10 and 11 are in identifier list mode.
	1	Filters 10 and 11 are in mask mode (filter 11 is the mask for the filter 10).
IFM[6]	0	Filters 12 and 13 are in identifier list mode.
	1	Filters 12 and 13 are in mask mode (filter 13 is the mask for the filter 12).

**Table 268. IFMR[IFM] configuration (continued)**

Bit	Value	Result
IFM[7]	0	Filters 14 and 15 are in identifier list mode.
	1	Filters 14 and 15 are in mask mode (filter 15 is the mask for the filter 14).

**24.10.20 Identifier filter control registers (IFCR0–IFCR15)**

The function of these registers is different depending on which mode the LINFlexD controller is in, as described in [Table 269](#).

**Table 269. IFCR functionality based on mode**

Mode	IFCR functionality
Identifier list	Each IFCR register acts as a filter.
Identifier mask	If $a = (\text{number of filters}) / 2$ , and $n = 0$ to $(a - 1)$ , then IFCR[2n] acts as a filter and IFCR[2n+1] acts as the mask for IFCR[2n].

*Note:* These registers are available on LINFlexD\_0 only.

Offsets: 0x4C–0x88 (16 registers)

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DFL <sup>1</sup>				DIR <sup>1</sup>		CCS <sup>1</sup>		0	0	ID <sup>1</sup>					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 269. Identifier filter control registers (IFCR0–IFCR15)**

**Table 270. IFCR field descriptions**

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame.
DIR	Direction This bit controls the direction of the data field. 0: LINFlexD receives the data and copy them in the BDRL and BDRM registers. 1: LINFlexD transmits the data from the BDRL and BDRM registers.



**Table 270. IFCR field descriptions (continued)**

Field	Description
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0: Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1: Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and below.
ID	Identifier Identifier part of the identifier field without the identifier parity.

**24.10.21 Global control register (GCR)**

This register can be programmed only in Initialization mode. The configuration specified in this register applies in both LIN and UART modes.

Offset: 0x8C (for LINFlexD\_0 only), 0x4C (for LINFlexD\_1)

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	TDFBM <sup>1</sup>	RDFBM <sup>1</sup>	TDLIS <sup>1</sup>	RDLIS <sup>1</sup>	STOP <sup>1</sup>	0
W																SR <sup>1</sup>
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This field is writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 270. Global control register (GCR)**

**Table 271. GCR field descriptions**

Field	Description
TDFBM	Transmit data first bit MSB This field controls the first bit of transmitted data (payload only) as MSB/LSB in both UART and LIN modes. 0 The first bit of transmitted data is LSB – that is, the first bit transmitted is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)). 1 The first bit of transmitted data is MSB – that is, the first bit transmitted is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)).





**Table 271. GCR field descriptions (continued)**

Field	Description
RDFBM	Received data first bit MSB This field controls the first bit of received data (payload only) as MSB/LSB in both UART and LIN modes. 0 The first bit of received data is LSB – that is, the first bit received is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)). 1 The first bit of received data is MSB – that is, the first bit received is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)).
TDLIS	Transmit data level inversion selection This field controls the data inversion of transmitted data (payload only) in both UART and LIN modes. 0 Transmitted data is not inverted. 1 Transmitted data is inverted.
RDLIS	Received data level inversion selection This field controls the data inversion of received data (payload only) in both UART and LIN modes. 0 Received data is not inverted. 1 Received data is inverted.
STOP	Stop bit configuration This field controls the number of stop bits in transmitted data in both UART and LIN modes. The stop bit is configured for all the fields (delimiter, sync, ID, checksum, and payload). 0 One stop bit 1 Two stop bits
SR	Soft reset If the software writes a “1” to this field, the LINFlexD controller executes a soft reset in which the FSMs, FIFO pointers, counters, timers, status registers, and error registers are reset but the configuration registers are unaffected. This field always reads “0”.

**24.10.22 UART preset timeout register (UARTPTO)**

This register contains the preset timeout value in UART mode, and is used to monitor the IDLE state of the reception line. The timeout detection uses this register and the UARTCTO register described in [Section 24.10.23: UART current timeout register \(UARTCTO\)](#).

Offset: 0x90 (for LINFlexD\_0 only), 0x50 (for LINFlexD\_1) Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PTO											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure 271. UART preset timeout register (UARTPTO)

Table 272. UARTPTO field descriptions

Field	Description
PTO	Preset value of the timeout counter Do not set PTO = 0 (otherwise, UARTSR[TO] would immediately be set).

### 24.10.23 UART current timeout register (UARTCTO)

This register contains the current timeout value in UART mode, and is used in conjunction with the UARTPTO register (see [Section 24.10.22: UART preset timeout register \(UARTPTO\)](#)) to monitor the IDLE state of the reception line. UART timeout works in both CPU and DMA modes.

The timeout counter:

- Starts at zero and counts upward
- Is clocked with the baud rate clock prescaled by a hard-wired scaling factor of 16
- Is automatically enabled when UARTCR[RXEN] = 1

Offset: 0x94 (for LINFlexD\_0 only), 0x54 (for LINFlexD\_1) Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	CTO											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 272. UART current timeout register (UARTCTO)

**Table 273. UARTCTO field descriptions**

Field	Description
CTO	<p>Current value of the timeout counter</p> <p>This field is reset whenever one of the following occurs:</p> <ul style="list-style-type: none"> <li>– A new value is written to the UARTPTO register</li> <li>– The value of this field matches the value of UARTPTO[PTO]</li> <li>– A hard or soft reset occurs</li> <li>– New incoming data is received</li> </ul> <p>When CTO matches the value of UARTPTO[PTO], UARTSR[TO] is set.</p>

**24.10.24 DMA Tx enable register (DMATXE)**

This register enables the DMA Tx interface.

Offset: 0x98 (for LINFlexD\_0 only), 0x58 (for LINFlexD\_1) Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTE15	DTE14	DTE13	DTE12	DTE11	DTE10	DTE9	DTE8	DTE7	DTE6	DTE5	DTE4	DTE3	DTE2	DTE1	DTE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 273. DMA Tx enable register (DMATXE)**

**Table 274. DMATXE field descriptions**

Field	Description
DTE <sub>n</sub>	<p>DMA Tx channel <i>n</i> enable</p> <p>0 DMA Tx channel <i>n</i> disabled</p> <p>1 DMA Tx channel <i>n</i> enabled</p> <p><b>Note:</b> When DMATXE = 0x0, the DMA Tx interface FSM is forced (soft reset) into the IDLE state.</p>

**24.10.25 DMA Rx enable register (DMARXE)**

This register enables the DMA Rx interface.

Offset: 0x9C (for LINFlexD\_0 only), 0x5C (for LINFlexD\_1)

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DRE15	DRE14	DRE13	DRE12	DRE11	DRE10	DRE9	DRE8	DRE7	DRE6	DRE5	DRE4	DRE3	DRE2	DRE1	DRE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 274. DMA Rx enable register (DMARXE)

Table 275. DMARXE field descriptions

Field	Description
DRE <sub>n</sub>	DMA Rx channel <i>n</i> enable 0 DMA Rx channel <i>n</i> disabled 1 DMA Rx channel <i>n</i> enabled  <b>Note:</b> When DMARXE = 0x0, the DMA Rx interface FSM is forced (soft reset) into the IDLE state.

## 24.11 DMA interface

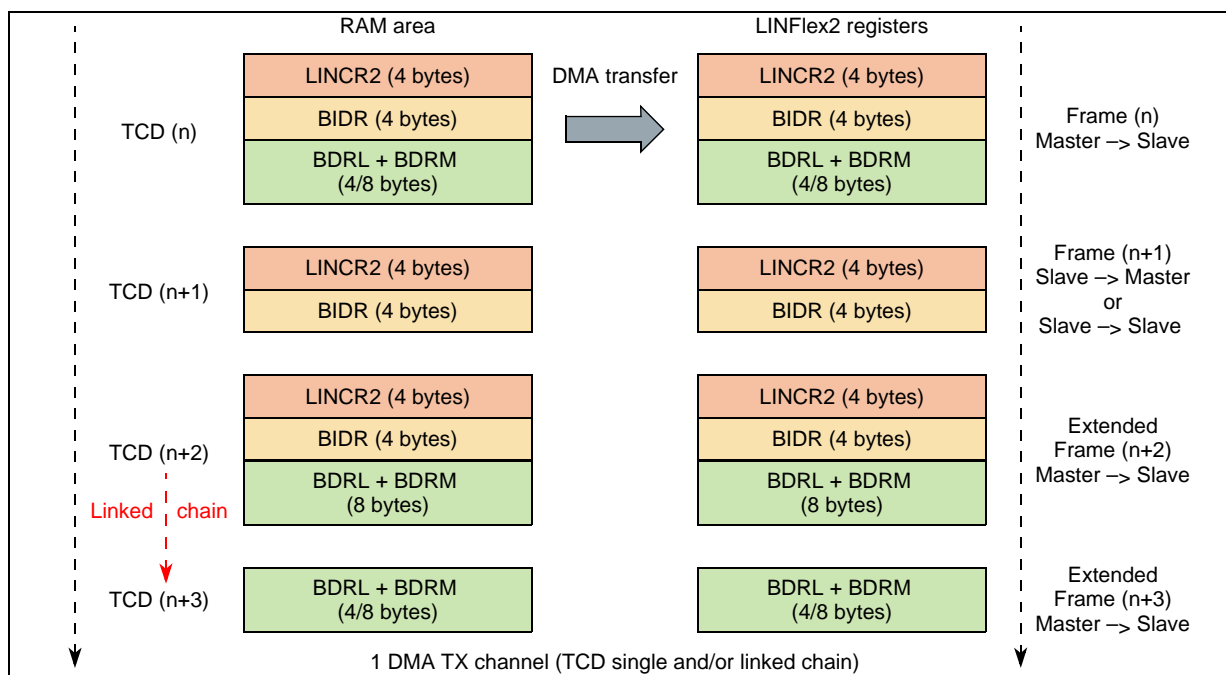
The LINFlexD DMA interface offers a parametric and programmable solution with the following features:

- LIN Master node, TX mode: single DMA channel
- LIN Master node, RX mode: single DMA channel
- LIN Slave node, TX mode: 1 to N DMA channels where N = max number of ID filters
- LIN Slave node, RX mode: 1 to N DMA channels where N = max number of ID filters
- UART node, TX mode: single DMA channel
- UART node, RX mode: single DMA channel + timeout

The LINFlexD controller interacts with an enhanced direct memory access (eDMA) controller; see the description of that controller for details on its operation and the transfer control descriptors (TCDs) referenced in this section.

### 24.11.1 Master node, TX mode

On a master node in TX mode, the DMA interface requires a single TX channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in [Figure 275](#).



**Figure 275. TCD chain memory map (master node, TX mode)**

The TCD chain of the DMA Tx channel on a master node supports:

- Master to Slave: transmission of the entire frame (header + data)
- Slave to Master: transmission of the header. The data reception is controlled by the Rx channel on the master node.
- Slave to Slave: transmission of the header.

The register settings for the LINC2 and BDR registers for each class of LIN frame are shown in [Table 276](#).

**Table 276. Register settings (master node, TX mode)**

LIN frame	LINC2	BDR
Master to Slave	DDRQ=1 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 1 (TX)
Slave to Master	DDRQ=0 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 0 (RX)
Slave to Slave	DDRQ=1 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 0 (RX)

The concept FSM to control the DMA TX interface is shown in [Figure 276](#). The DMA TX FSM will move to IDLE state immediately at next clock edge if DMATXE[0] = 0.

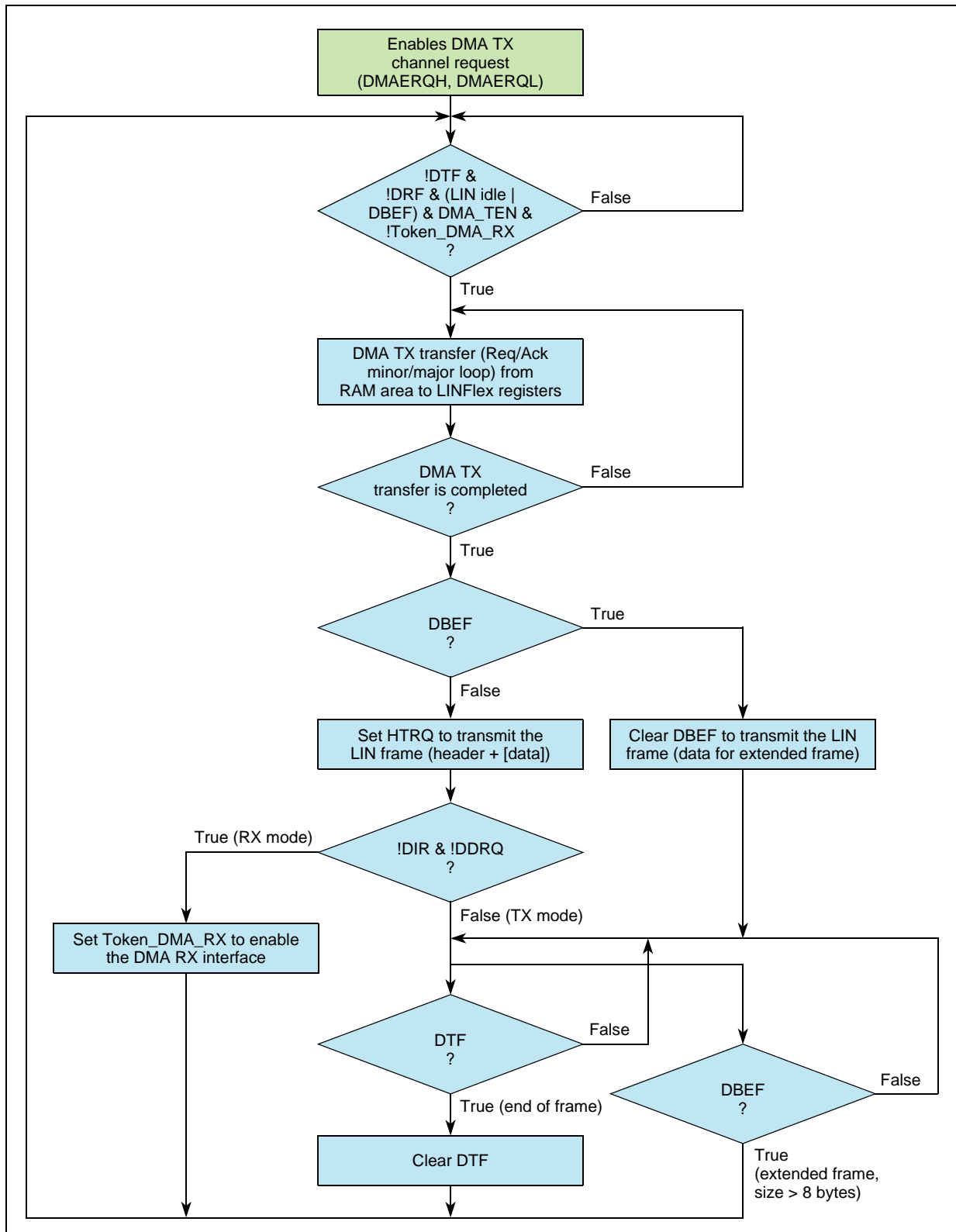


Figure 276. FSM to control the DMA TX interface (master node)

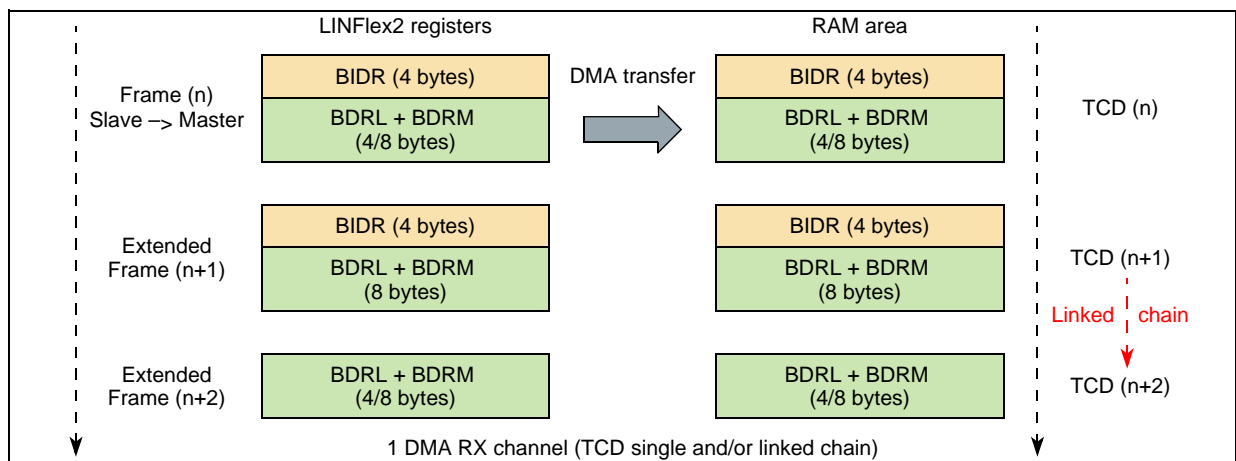
The TCD settings (word transfer) are shown in [Table 277](#). All other TCD fields are equal to 0. TCD settings based on half-word or byte transfers are allowed.

**Table 277. TCD settings (master node, TX mode)**

TCD field	Value	Description
CITER[14:0]	1	Single iteration for the “major” loop
BITER[14:0]	1	Single iteration for the “major” loop
NBYTES[31:0]	$[4 + 4] + 0/4/8 = N$	Data buffer is stuffed with dummy bytes if the length is not word aligned. LINCR2 + BIDR + BDRL + BDRM
SADDR[31:0]	RAM address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	-N	
DADDR[31:0]	LINCR2 address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	-N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

### 24.11.2 Master node, RX mode

On a master node in RX mode, the DMA interface requires a single RX channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in [Figure 277](#).

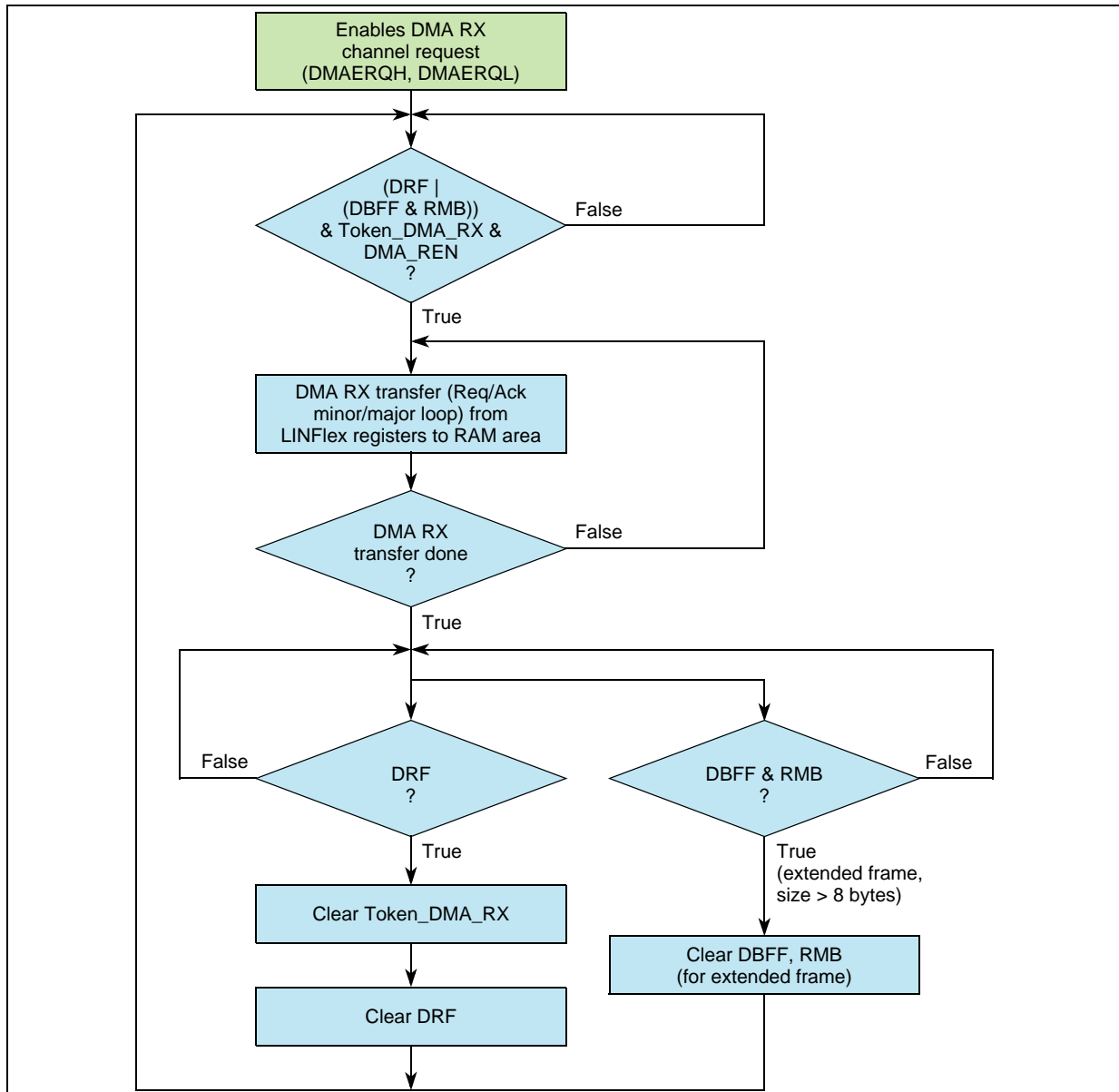


**Figure 277. TCD chain memory map (master node, RX mode)**

The TCD chain of the DMA Rx channel on a master node supports Slave-to-Master reception of the data field.

The BIDR register is optionally copied into the RAM area. This BIDR field (part of FIFO data) contains the ID of each message to allow the CPU to figure out which ID was received by the LINFlexD DMA if only the “one DMA channel” setup is used.

The concept FSM to control the DMA RX interface is shown in *Figure 278*. The DMA RX FSM will move to IDLE state immediately at next clock edge if DMARXE[0]=0.



**Figure 278. FSM to control the DMA RX interface (master node)**

The TCD settings (word transfer) are shown in *Table 278*. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.



Table 278. TCD settings (master node, RX mode)

TCD field	Value	Description
CITER[14:0]	1	Single iteration for the “major” loop
BITER[14:0]	1	Single iteration for the “major” loop
NBYTES[31:0]	$[4] + 4/8 = N$	Data buffer is stuffed with dummy bytes if the length is not word aligned. BIDR + BDRL + BDRM
SADDR[31:0]	BIDR address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	-N	
DADDR[31:0]	RAM address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	-N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

### 24.11.3 Slave node, TX mode

On a slave node in TX mode, the DMA interface requires a DMA TX channel for each ID filter programmed in TX mode. In case a single DMA TX channel is available, a single ID field filter must be programmed in TX mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in [Figure 279](#).

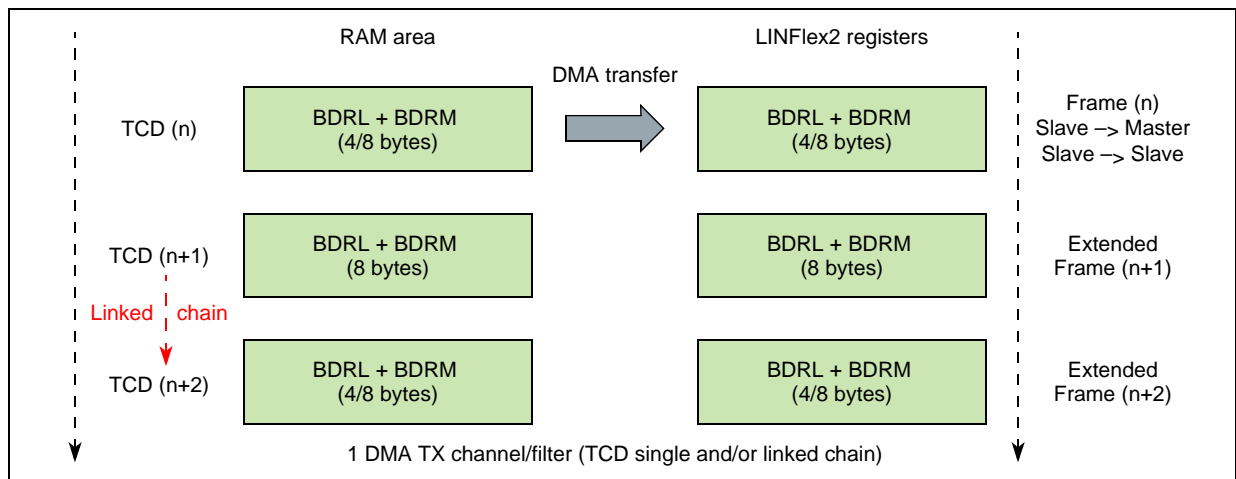


Figure 279. TCD chain memory map (slave node, TX mode)

The TCD chain of the DMA Tx channel on a slave node supports:

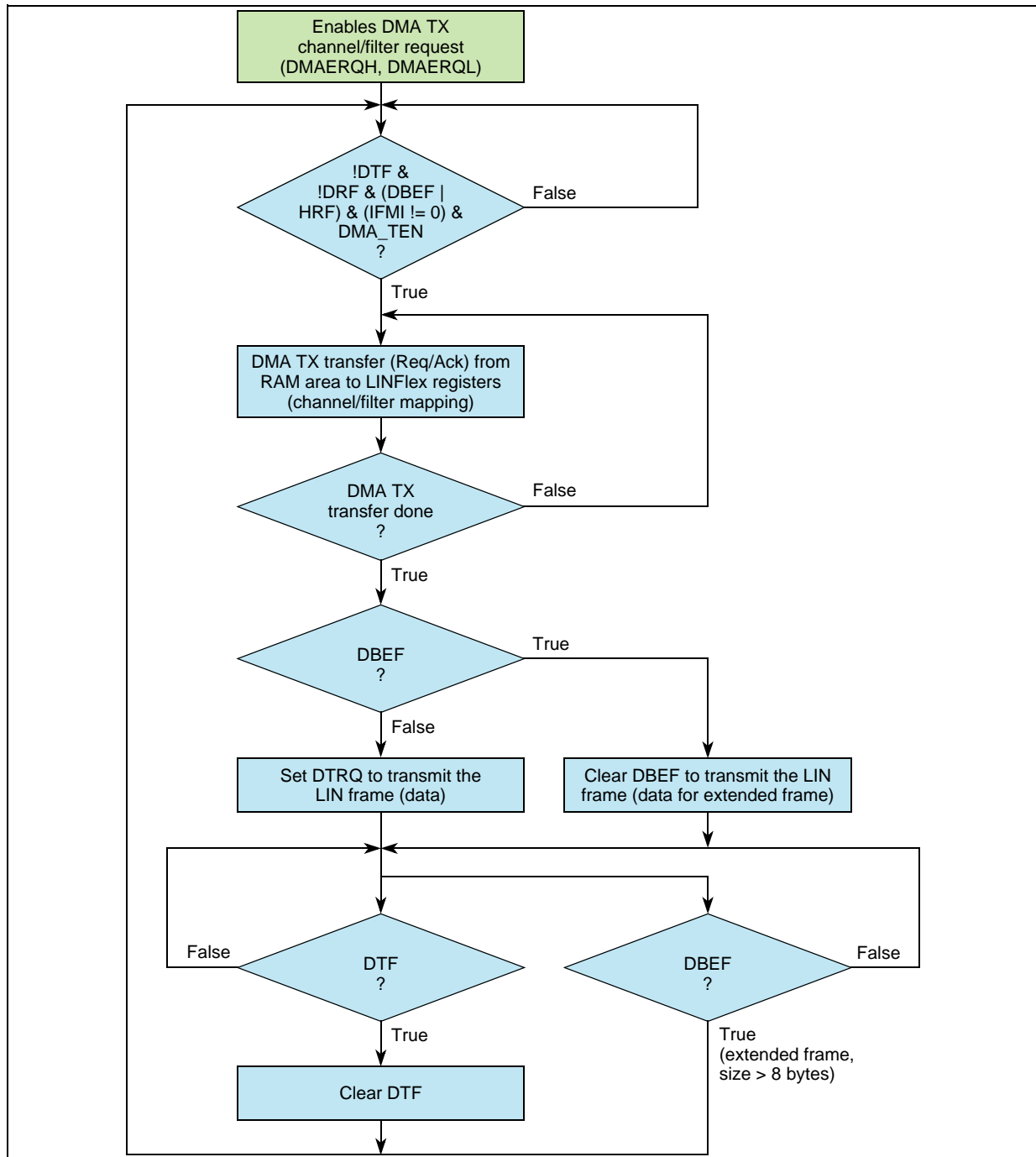
- Slave to Master: transmission of the data field
- Slave to Slave: transmission of the data field

The register settings of the LINCR2, IFER, IFMR, and IFCR registers are shown in [Table 279](#).

**Table 279. Register settings (slave node, TX mode)**

LIN frame	LINCR2	IFER	IFMR	IFCR
Slave to Master or Slave to Slave	DDRQ = 0 DTRQ = 0 HTRQ = 0	To enable an ID filter (Tx mode) for each DMA TX channel	- Identifier list mode - Identifier mask mode	DFL = payload size ID = address CCS = checksum DIR = 1(TX)

The concept FSM to control the DMA Tx interface is shown in [Figure 280](#). DMA TX FSM will move to idle state if  $DMATXE[x] = 0$ , where  $x = IFMI - 1$ .



**Figure 280. FSM to control the DMA TX interface (slave node)**

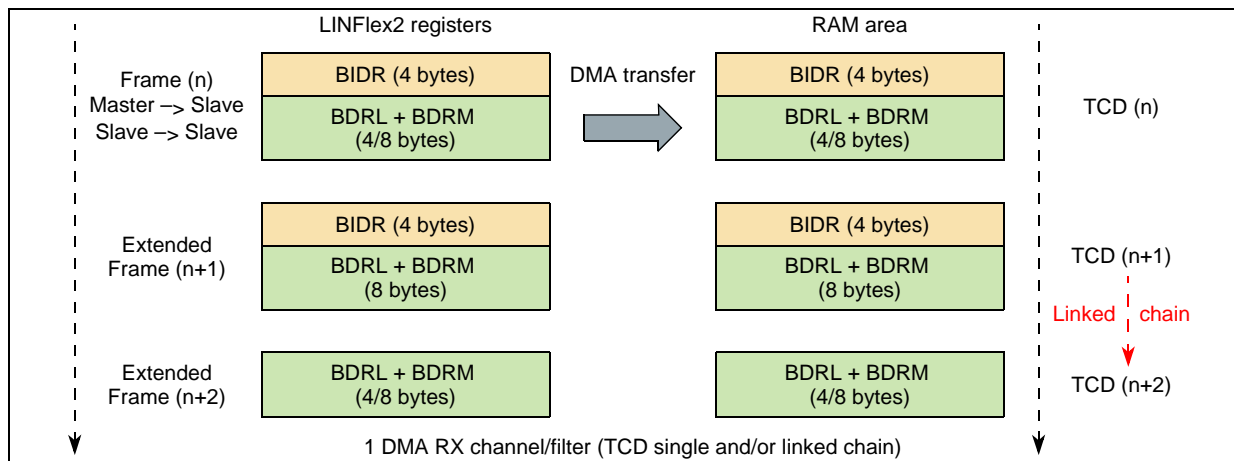
The TCD settings (word transfer) are shown in [Table 280](#). All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.

**Table 280. TCD settings (slave node, TX mode)**

TCD field	Value	Description
CITER[14:0]	1	Single iteration for the “major” loop
BITER[14:0]	1	Single iteration for the “major” loop
NBYTES[31:0]	4/8 = N	Data buffer is stuffed with dummy bytes if the length is not word aligned. BDRL + BDRM
SADDR[31:0]	RAM address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	-N	
DADDR[31:0]	BDRL address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	-N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

**24.11.4 Slave node, RX mode**

On a slave node in RX mode, the DMA interface requires a DMA RX channel for each ID filter programmed in RX mode. In case a single DMA RX channel is available, a single ID field filter must be programmed in RX mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in [Figure 281](#).



**Figure 281. TCD chain memory map (slave node, RX mode)**

The TCD chain of the DMA RX channel on a slave node supports:

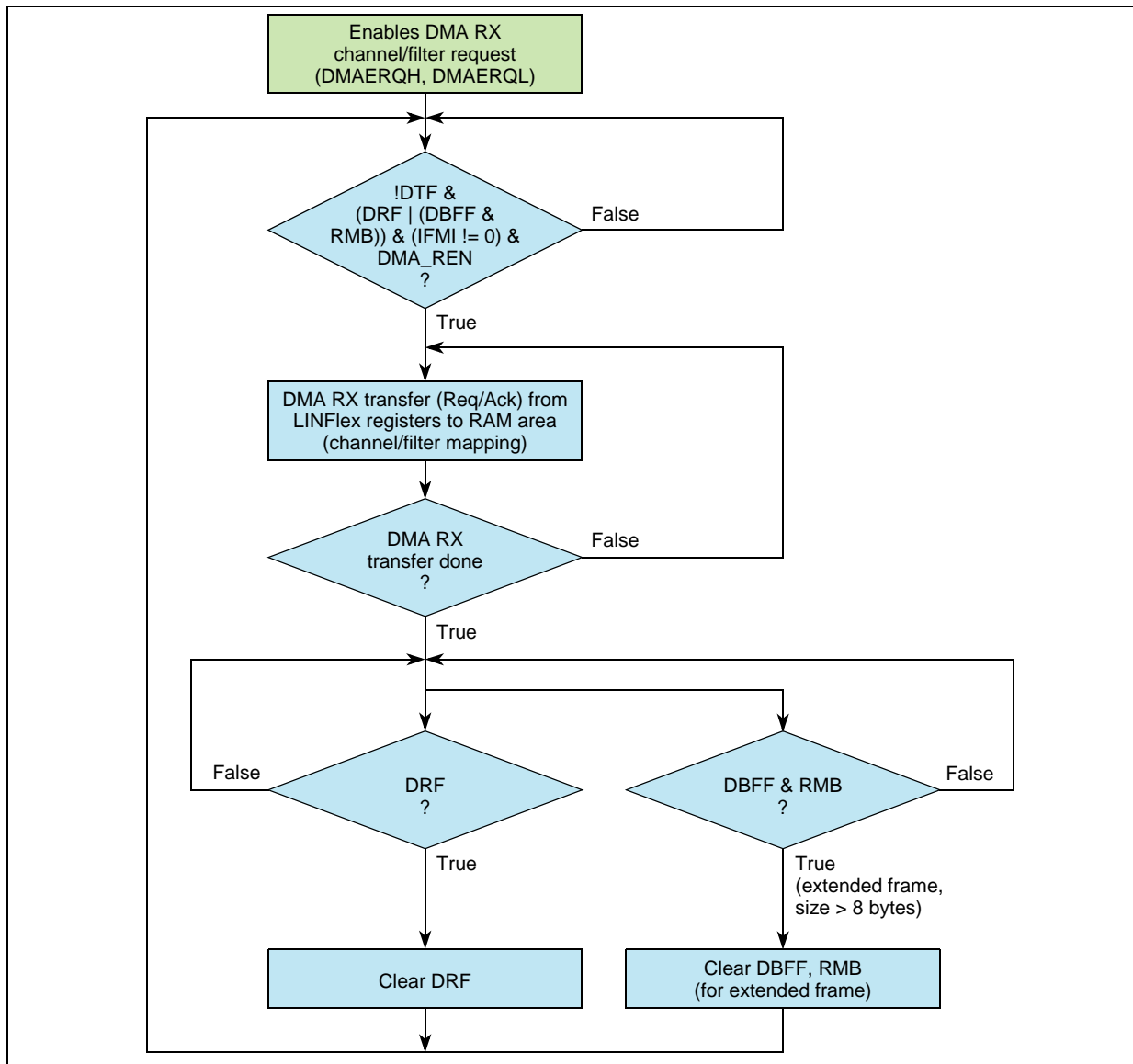
- Master to Slave: reception of the data field.
- Slave to Slave: reception of the data field.

The register setting of the LINC2, IFER, IFMR, and IFCR registers are given in [Table 281](#).

**Table 281. Register settings (slave node, RX mode)**

LIN frame	LINC2	IFER	IFMR	IFCR
Master to Slave or Slave to Slave	DDRQ = 0 DTRQ = 0 HTRQ = 0	To enable an ID filter (Rx mode) for each DMA RX channel	- Identifier list mode - Identifier mask mode	DFL = payload size ID = address CCS = checksum DIR = 0 (RX)

The concept FSM to control the DMA Rx interface is shown in [Figure 282](#). DMA RX FSM will move to idle state if DMARXE[x] = 0 where x = IFMI - 1.



**Figure 282. FSM to control the DMA RX interface (slave node)**

The TCD settings (word transfer) are shown in [Table 282](#). All other TCD fields = 0. TCD settings based on half-word or byte transfer are allowed.

**Table 282. TCD settings (slave node, RX mode)**

TCD Field	Value	Description
CITER[14:0]	1	Single iteration for the "major" loop
BITER[14:0]	1	Single iteration for the "major" loop
NBYTES[31:0]	[4] + 4/8 = N	Data buffer is stuffed with dummy bytes if the length is not word aligned. BIDR + BDRL + BDRM
SADDR[31:0]	BDRL address	

Table 282. TCD settings (slave node, RX mode) (continued)

TCD Field	Value	Description
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	-N	
DADDR[31:0]	RAM address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	-N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain.
START	0	No software request

### 24.11.5 UART node, TX mode

In UART TX mode, the DMA interface requires a DMA TX channel. A single TCD can control the transmission of an entire Tx buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in [Figure 283](#).

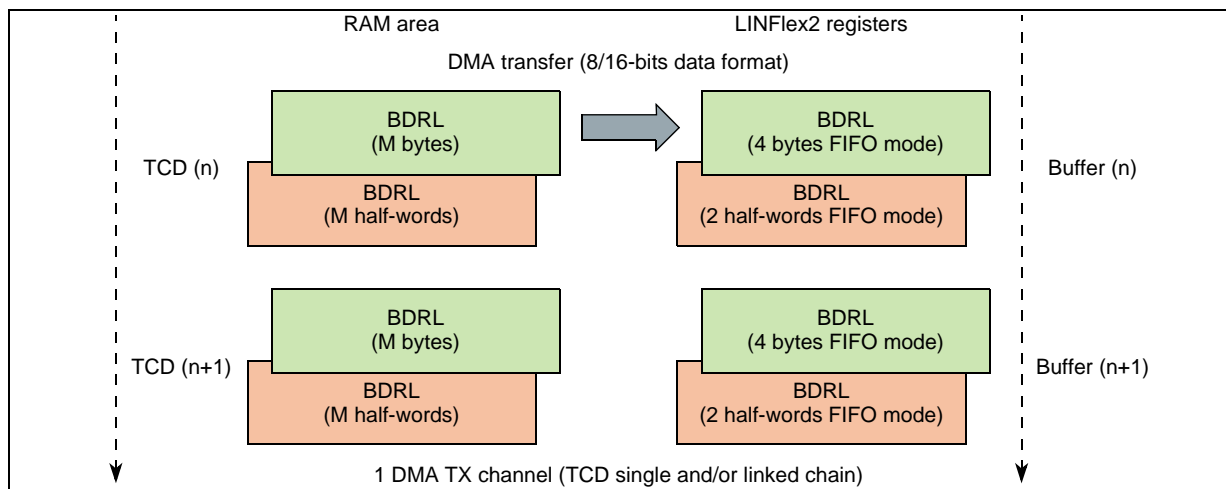


Figure 283. TCD chain memory map (UART node, TX mode)

The UART TX buffer must be configured in FIFO mode in order to:

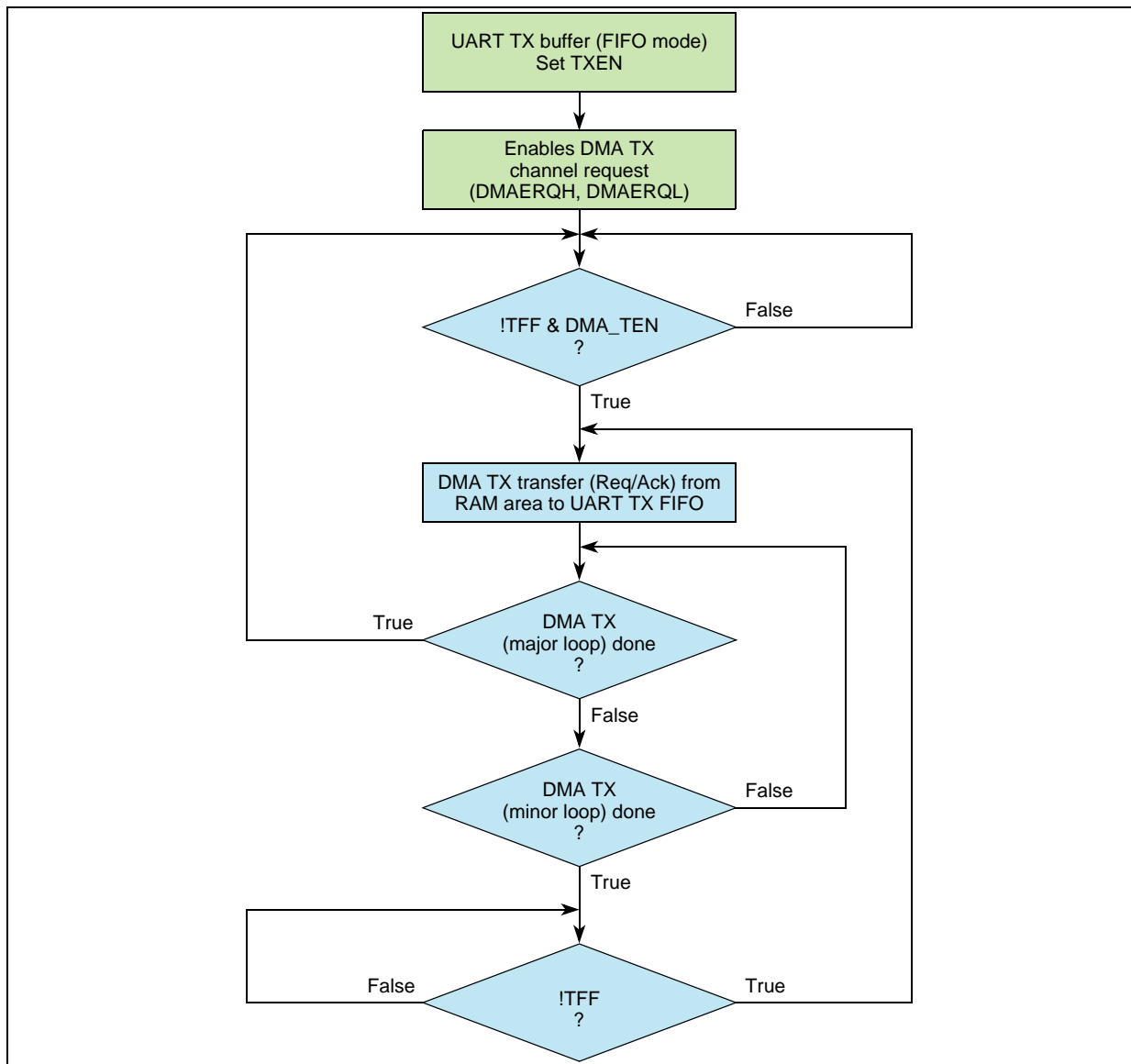
- Allow the transfer of large data buffer by a single TCD
- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from the RAM to the FIFO
- Use low priority DMA channels
- Support the UART baud rate (2 Mb/s) without underrun events

The Tx FIFO size is:

- 4 bytes in 8-bit data format
- 2 half-words in 16-bit data format

A DMA request is triggered by FIFO not full (TX) status signals.

The concept FSM to control the DMA TX interface is shown in [Figure 284](#). DMA TX FSM will move to idle state if DMATXE[0] = 0.



**Figure 284. FSM to control the DMA TX interface (UART node)**

The TCD settings (typical case) are shown in [Table 283](#). All other TCD fields = 0. The minor loop transfers a single byte/half-word as soon a free entry is available in the Tx FIFO.



Table 283. TCD settings (UART node, TX mode)

TCD Field	Value		Description
	8-bit data	16-bit data	
CITER[14:0]	M		Multiple iterations for the “major” loop
BITER[14:0]	M		Multiple iterations for the “major” loop
NBYTES[31:0]	1	2	Minor loop transfer = 1 or 2 bytes
SADDR[31:0]	RAM address		
SOFF[15:0]	1	2	Byte/Half-word increment
SSIZE[2:0]	0	1	Byte/Half-word transfer
SLAST[31:0]	-M	-M * 2	
DADDR[31:0]	BDRL address		DADDR = BDRL + 0x3 for byte transfer DADDR = BDRL + 0x2 for half-word transfer
DOFF[15:0]	0		No increment (FIFO)
DSIZE[2:0]	0	1	Byte/Half-word transfer
DLAST_SGA[31:0]	0		No scatter/gather processing
INT_MAJ	0/1		Interrupt disabled/enabled
D_REQ	1		Only on the last TCD of the chain.
START	0		No software request

### 24.11.6 UART node, RX mode

In UART RX mode, the DMA interface requires a DMA RX channel. A single TCD can control the reception of an entire Rx buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in [Figure 285](#).

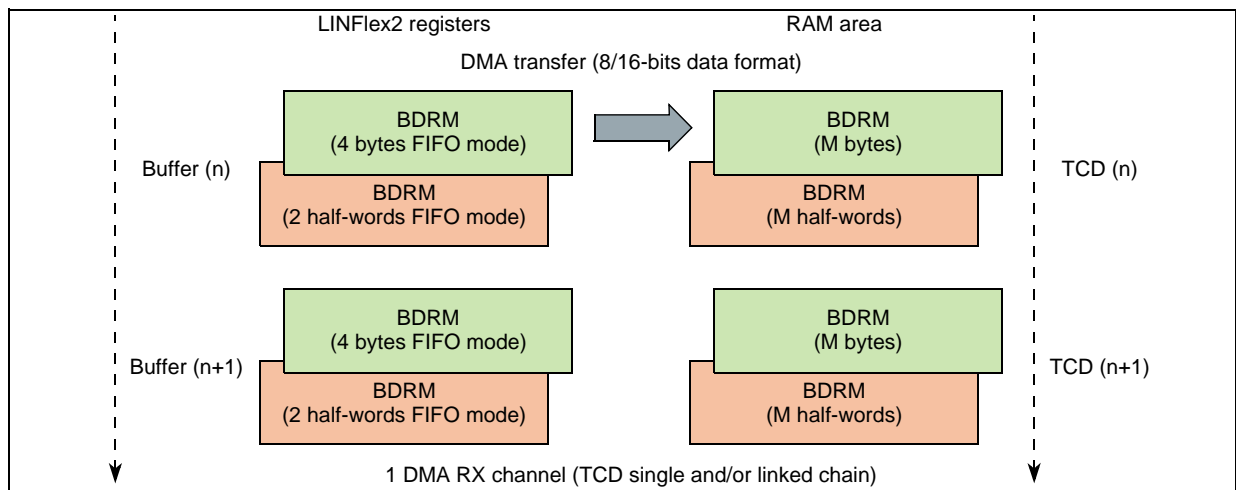


Figure 285. TCD chain memory map (UART node, RX mode)

The UART RX buffer must be configured in FIFO mode in order to:

- Allow the transfer of large data buffer by a single TCD
- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from the FIFO to the RAM
- Use low priority DMA channels
- Support high UART baud rate (at least 2 Mb/s) without overrun events

The Rx FIFO size is:

- 4 bytes in 8-bit data format
- 2 half-words in 16-bit data format

This is sufficient because just one byte allows a reaction time of about 3.8  $\mu\text{s}$  (at 2 Mbit/s), corresponding to about 450 clock cycles at 120 MHz, before the transmission is affected. A DMA request is triggered by FIFO not empty (RX) status signals.

The concept FSM to control the DMA Rx interface is shown in [Figure 286](#). DMA Rx FSM will move to idle state if DMARXE[0] = 0.

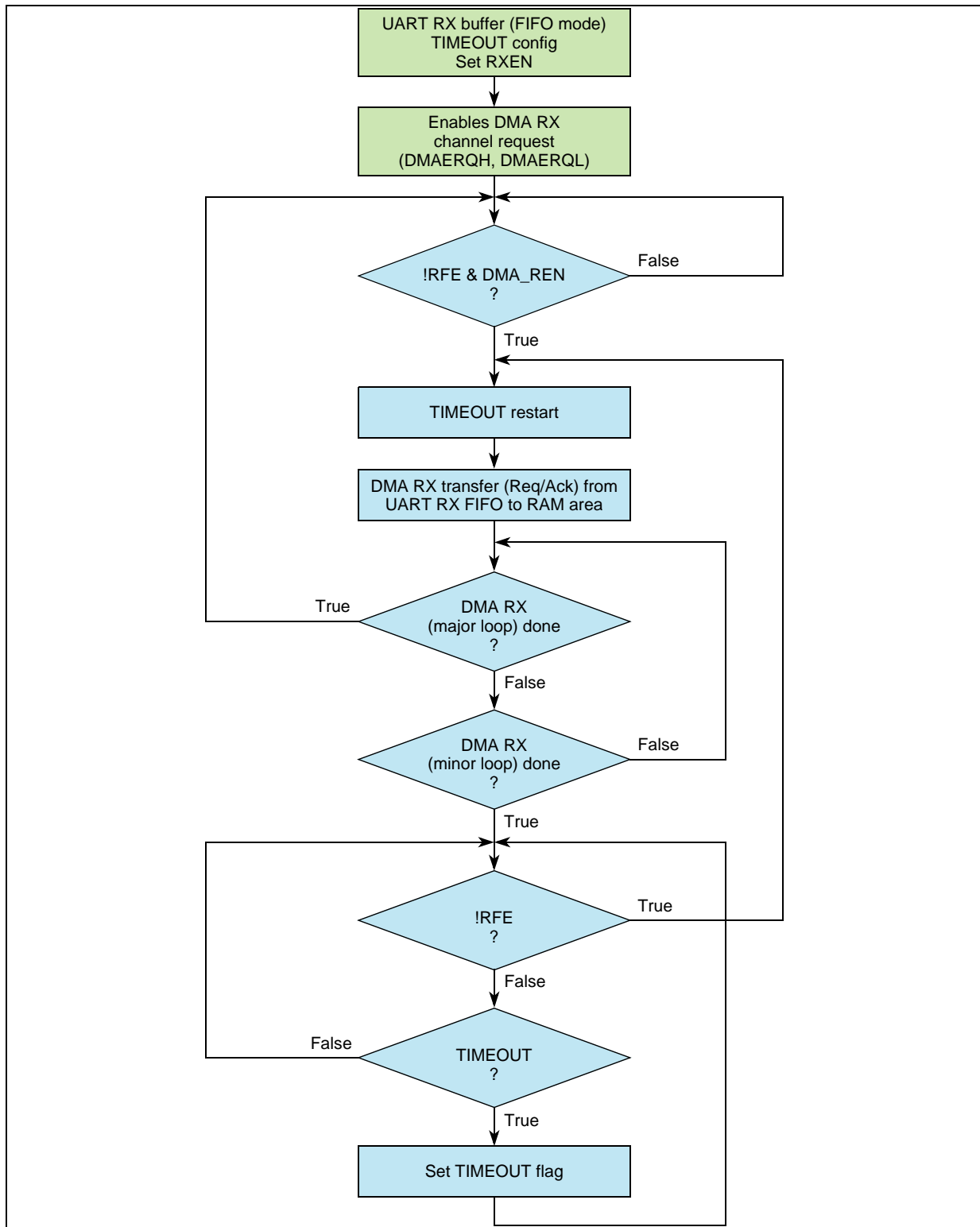


Figure 286. FSM to control the DMA RX interface (UART node)

The TCD settings (typical case) are shown in [Table 284](#). All other TCD fields = 0. The minor loop transfers a single byte/half-word as soon an entry is available in the Rx FIFO. A new software reset bit is required that allows the LINFlexD FSMs to be reset in case this timeout state is reached or in any other case. Timeout counter can be re-written by software at any time to extend timeout period.

**Table 284. TCD settings (UART node, RX mode)**

TCD Field	Value		Description
	8 bits data	16 bits data	
CITER[14:0]	M		Multiple iterations for the “major” loop
BITER[14:0]	M		Multiple iterations for the “major” loop
NBYTES[31:0]	1	2	Minor loop transfer = 1 or 2 bytes
SADDR[31:0]	BDRM address		SADDR = BDRM + 0x3 for byte transfer SADDR = BDRM + 0x2 for half-word transfer
SOFF[15:0]	0		No increment (FIFO)
SSIZE[2:0]	0	1	Byte/Half-word transfer
SLAST[31:0]	0		
DADDR[31:0]	RAM address		
DOFF[15:0]	1	2	Byte/Half-word increment
DSIZE[2:0]	0	1	Byte/Half-word transfer
DLAST_SGA[31:0]	-M	-M * 2	No scatter/gather processing
INT_MAJ	0/1		Interrupt disabled/enabled
D_REQ	1		Only on the last TCD of the chain.
START	0		No software request

**24.11.7 Use cases and limitations**

- In LIN slave mode, the DMA capability can be used only if the ID filtering mode is activated. The number of ID filters enabled must be equal to the number of DMA channels enabled. The correspondence between channel # and ID filter is based on IFMI (identifier filter match index).
- In LIN master mode both the DMA channels (TX and RX) must be enabled in case the DMA capability is required.
- In UART mode the DMA capability can be used only if the UART Tx/Rx buffers are configured as FIFOs.
- DMA and CPU operating modes are mutually exclusive for the data/frame transfer on a UART or LIN node. Once a DMA transfer is finished the CPU can handle subsequent accesses.
- Error management must be always executed via CPU enabling the related error interrupt sources. The DMA capability does not provide support for the error

management. Error management means checking status bits, handling IRQs and potentially canceling DMA transfers.

- The DMA programming model must be coherent with the TCD setting defined in this document.

## 24.12 Functional description

### 24.12.1 8-bit timeout counter

#### 24.12.1.1 LIN timeout mode

Resetting the LTOM bit in the LINTCSR enables the LIN timeout mode. The LINOOCR becomes read-only, and OC1 and OC2 output compare values in the LINOOCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the MME bit in LINCR1), the 8-bit timeout counter will behave differently.

LIN timeout mode must not be enabled during LIN extended frames transmission or reception (that is, if the data field length in the BIDR is configured with a value higher than 8 data bytes).

##### 24.12.1.1.1 LIN Master mode

Field RTO in the LINTOCR can be used to tune response timeout and frame timeout values. Header timeout value is fixed to  $HTO = 28$ -bit time.

Field OC1 checks  $T_{Header}$  and  $T_{Response}$  and field OC2 checks  $T_{Frame}$  (refer to [Figure 287](#)).

When LINFlexD moves from Break delimiter state to Synch Field state (refer to [Section 24.10.3: LIN status register \(LINSR\)](#)):

- OC1 is updated with the value of  $OC_{Header}$  ( $OC_{Header} = CNT + 28$ ),
- OC2 is updated with the value of  $OC_{Frame}$  ( $OC_{Frame} = CNT + 28 + RTO \times 9$  (frame timeout value for an 8-byte frame)),
- the TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of  $OC_{Response}$  ( $OC_{Response} = CNT + RTO \times 9$  (response timeout value for an 8-byte frame)).

On the first response byte is received, OC1 and OC2 are automatically updated to check  $T_{Response}$  and  $T_{Frame}$  according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or response, the TOCE bit is reset.

If there is no response, frame timeout value does not take into account the DFL value, and an 8-byte response ( $DFL = 7$ ) is always assumed.

##### 24.12.1.1.2 LIN Slave mode

Field RTO in the LINTOCR can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO.

OC1 checks  $T_{Header}$  and  $T_{Response}$  and OC2 checks  $T_{Frame}$  (refer to [Figure 287](#)).

When LINFlexD moves from Break state to Break Delimiter state (refer to [Section 24.10.3: LIN status register \(LINSR\)](#)):

- OC1 is updated with the value of  $OC_{Header}$  ( $OC_{Header} = CNT + HTO$ ),
- OC2 is updated with the value of  $OC_{Frame}$  ( $OC_{Frame} = CNT + HTO + RTO \times 9$  (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of  $OC_{Response}$  ( $OC_{Response} = CNT + RTO \times 9$  (response timeout value for an 8-byte frame)).

Once the first response byte is received, OC1 and OC2 are automatically updated to check  $T_{Response}$  and  $T_{Frame}$  according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or data field, the TOCE bit is reset.

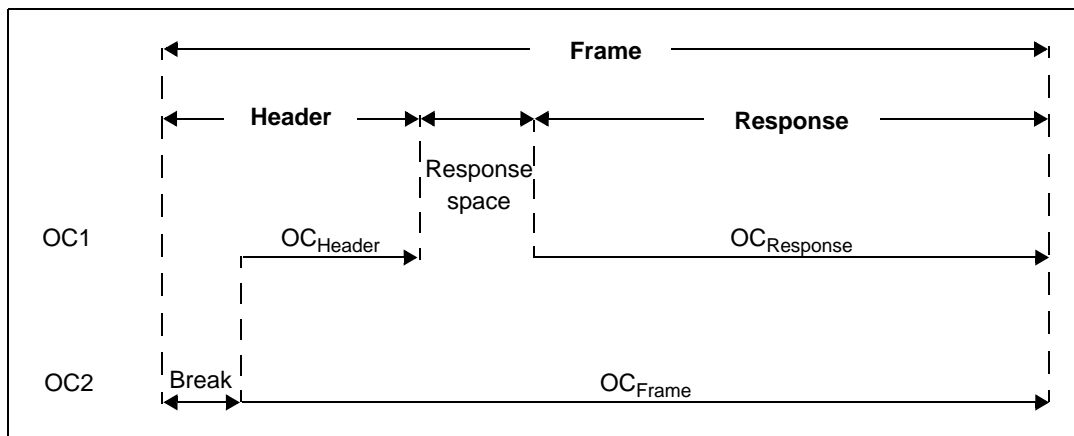


Figure 287. Header and response timeout

### 24.12.1.2 Output compare mode

Setting the LTOM bit in the LINTCSR enables the output compare mode. This mode allows the user to fully customize the use of the counter.

OC1 and OC2 output compare values can be updated in the LINTOCR by software.

### 24.12.2 Interrupts

Table 285. LINFlexD interrupt control

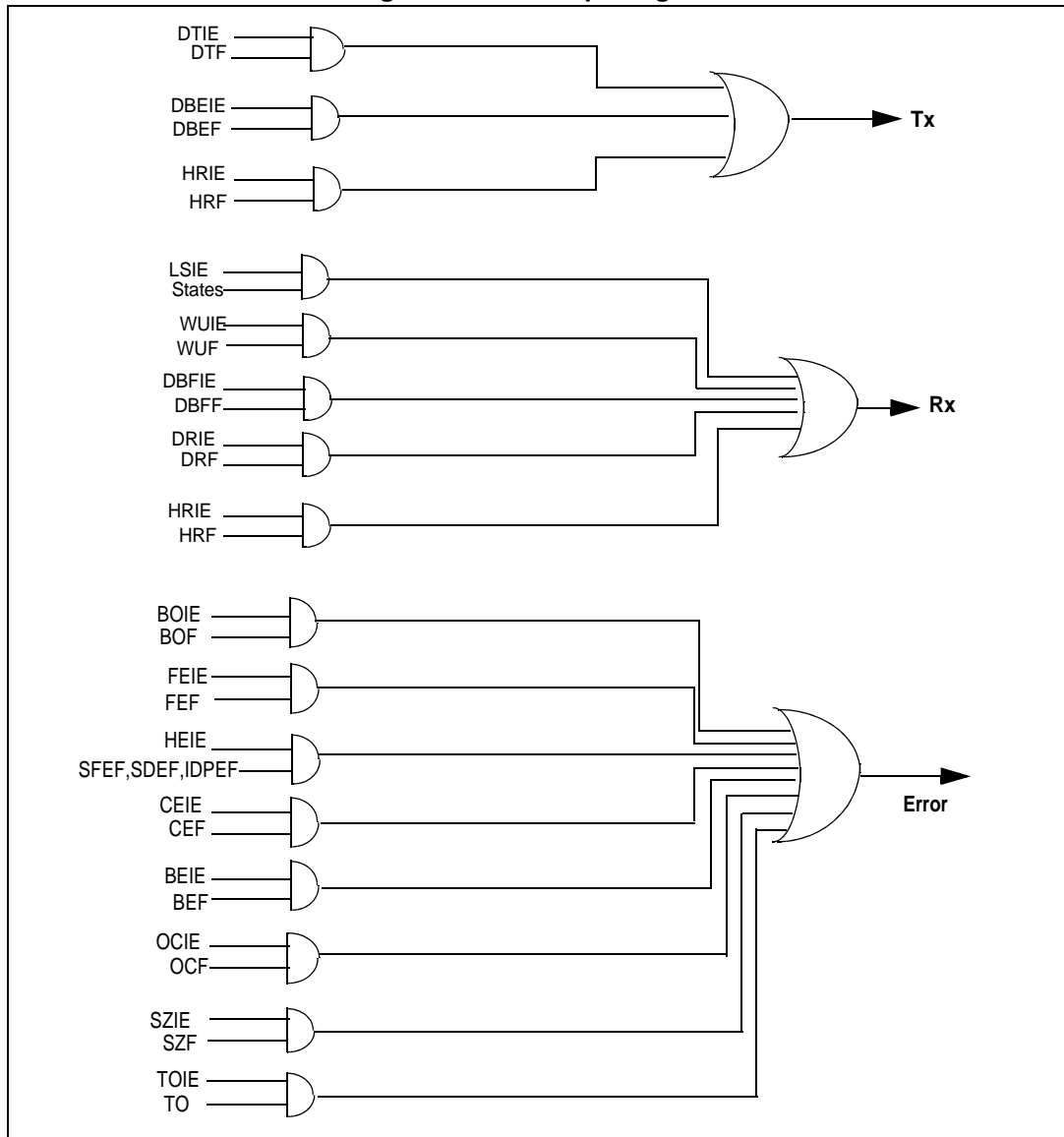
Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Header Received interrupt	HRF	HRIE	RXI <sup>(1)</sup>
Data Transmitted interrupt	DTF	DTIE	TXI
Data Received interrupt	DRF	DRIE	RXI
Data Buffer Empty interrupt	DBEF	DBEIE	TXI

Table 285. LINFlexD interrupt control (continued)

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Data Buffer Full interrupt	DBFF	DBFIE	RXI
Wake-up interrupt	WUPF	WUPIE	RXI
LIN State interrupt <sup>(2)</sup>	LSF	LSIE	RXI
Buffer Overrun interrupt	BOF	BOIE	ERR
Framing Error interrupt	FEF	FEIE	ERR
Header Error interrupt	HEF	HEIE	ERR
Checksum Error interrupt	CEF	CEIE	ERR
Bit Error interrupt	BEF	BEIE	ERR
Output Compare interrupt	OCF	OCIE	ERR
Stuck at Zero interrupt	SZF	SZIE	ERR

1. In Slave mode, if at least one filter is configured as TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.
2. For debug and validation purposes.

Figure 288. Interrupt diagram



### 24.12.3 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in the Mantissa (LINIBRR) and Fraction (LINFBR) registers.

$$\text{Tx/Rx baud} = \frac{f_{\text{ipg\_clock\_lin}}}{(16 * \text{LFDIV})}$$

LFDIV is an unsigned fixed point number. The 20-bit mantissa is coded in the LINIBRR register and the fraction is coded in the LINFBR register.

The following examples show how to derive LFDIV from LINIBRR and LINFBR register values:



**Example 6**

If LINIBRR = 27d and LINFBR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

**Example 7**

To program LFDIV = 25.62d,

LINFBR = 16 \* 0.62 = 9.92, nearest real number 10d = Ah

LINIBRR = mantissa(25.620d) = 25d = 19h

*Note: The Baud Counters are updated with the new value of the Baud Registers after a write to LINIBRR. Hence the Baud Register value must not be changed during a transaction. The LINFBR (containing the Fraction bits) must be programmed before LINIBRR.*

*Note: LFDIV must be greater than or equal to 1.5d, for example, LINIBRR = 1 and LINFBR = 8. Therefore, the maximum possible baudrate is f<sub>periph\_set1\_clk</sub> / 24.*

**Table 286. Error calculation for programmed baud rates**

Baud rate	f <sub>periph_set1_clk</sub> = 64 MHz				f <sub>periph_set1_clk</sub> = 16 MHz			
	Actual	Value programmed in the baud rate register		% Error = (Calculated - Desired) Baud rate / Desired baud rate	Actual	Value programmed in the baud rate register		% Error = (Calculated - Desired) Baud rate / Desired baud rate
		LINIBRR	LINFBR			LINIBRR	LINFBR	
2400	2399.97	1666	11	-0.001	2399.88	416	11	-0.005
9600	9599.52	416	11	-0.005	9598.08	104	3	-0.02
10417	10416.7	384	0	-0.003	10416.7	96	0	0
19200	19201.9	208	5	0.01	19207.7	52	1	0.04
57600	57605.8	69	7	0.01	57554	17	6	-0.08
115200	115108	34	12	-0.08	115108	8	11	-0.08
230400	230216	17	6	-0.08	231884	4	5	0.644
460800	460432	8	11	-0.08	457143	2	3	-0.794
921600	927536	4	5	0.644	941176	1	1	2.124

**24.13 Programming considerations**

This section describes the various configurations in which the LINFlexD can be used.



24.13.1 Master node

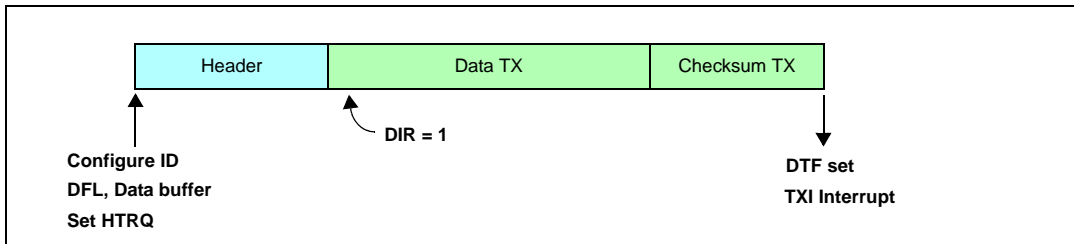


Figure 289. Programming consideration: master node, transmitter

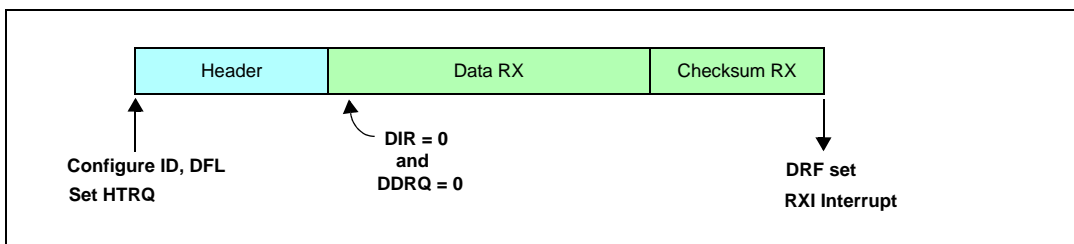


Figure 290. Programming consideration: master node, receiver

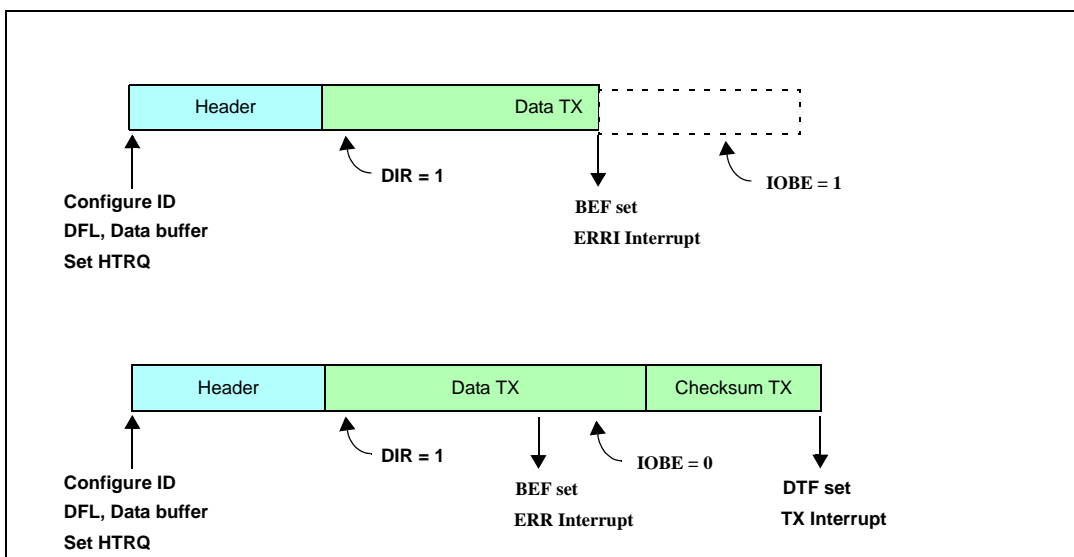


Figure 291. Programming consideration: master node, transmitter, bit error

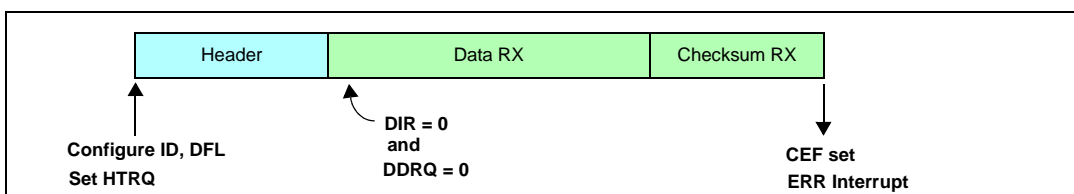


Figure 292. Programming consideration: master node, receiver, checksum error

24.13.2 Slave node

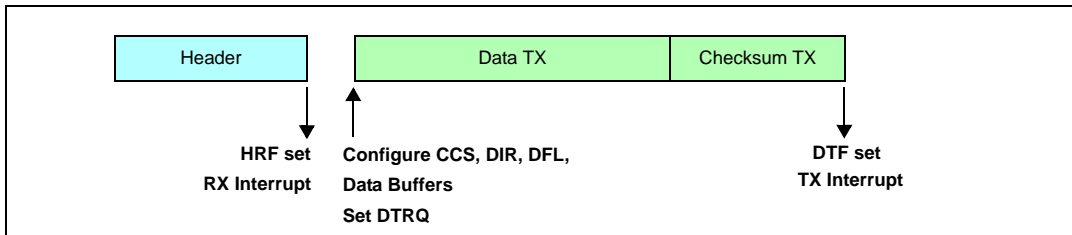


Figure 293. Programming consideration: slave node, transmitter, no filters

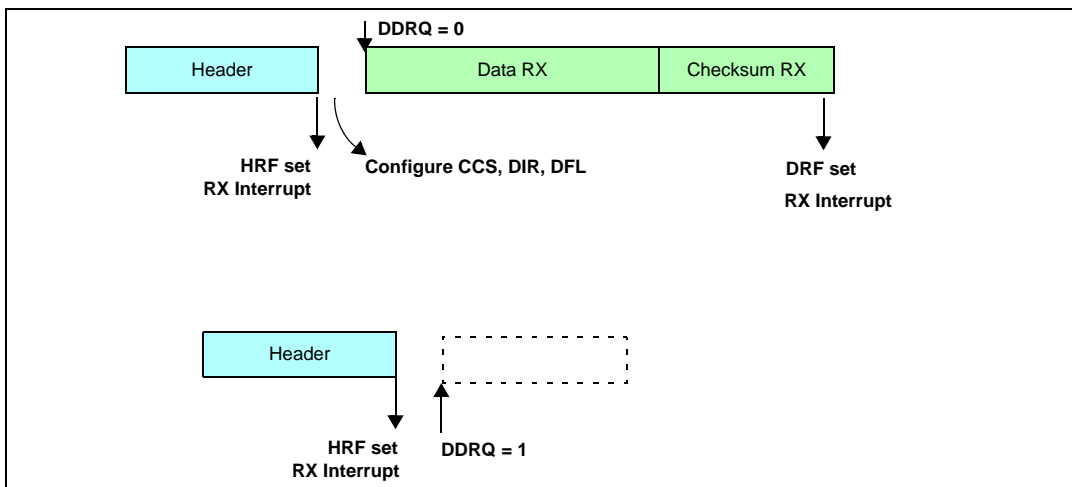


Figure 294. Programming consideration: slave node, receiver, no filters

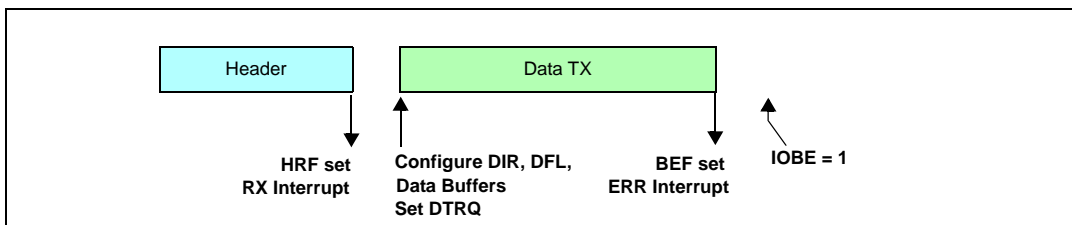


Figure 295. Programming consideration: slave node, transmitter, no filters, bit error

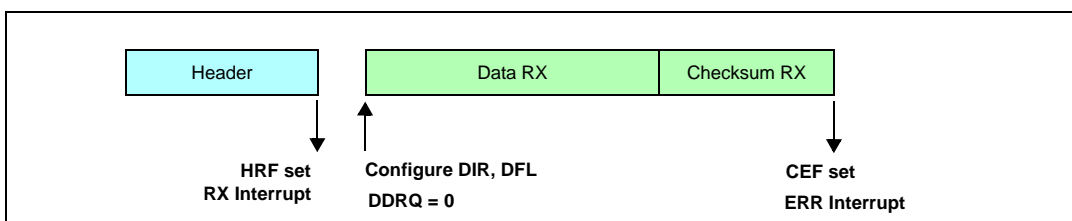
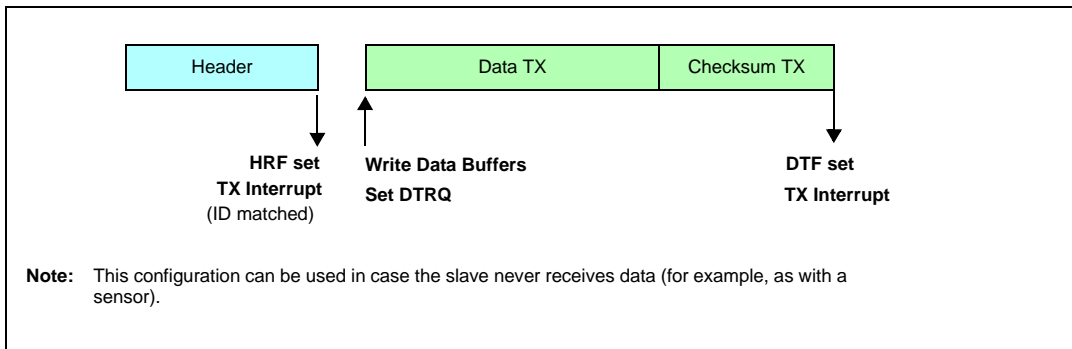
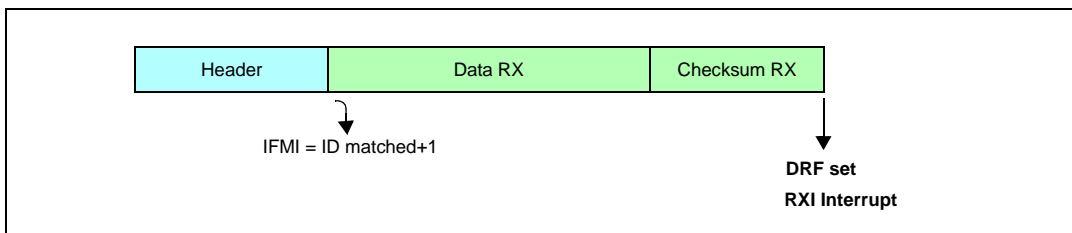


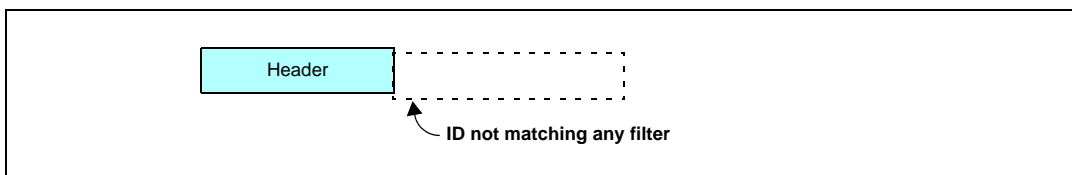
Figure 296. Programming consideration: slave node, receiver, no filters, checksum error



**Figure 297. Programming consideration: slave node, at least one TX filter, BF is reset, ID matches filter**



**Figure 298. Programming consideration: slave node, at least one RX filter, BF is reset, ID matches filter**



**Figure 299. Programming consideration: slave node, RX only, TX only, RX and TX filters, ID not matching filter, BF is reset**

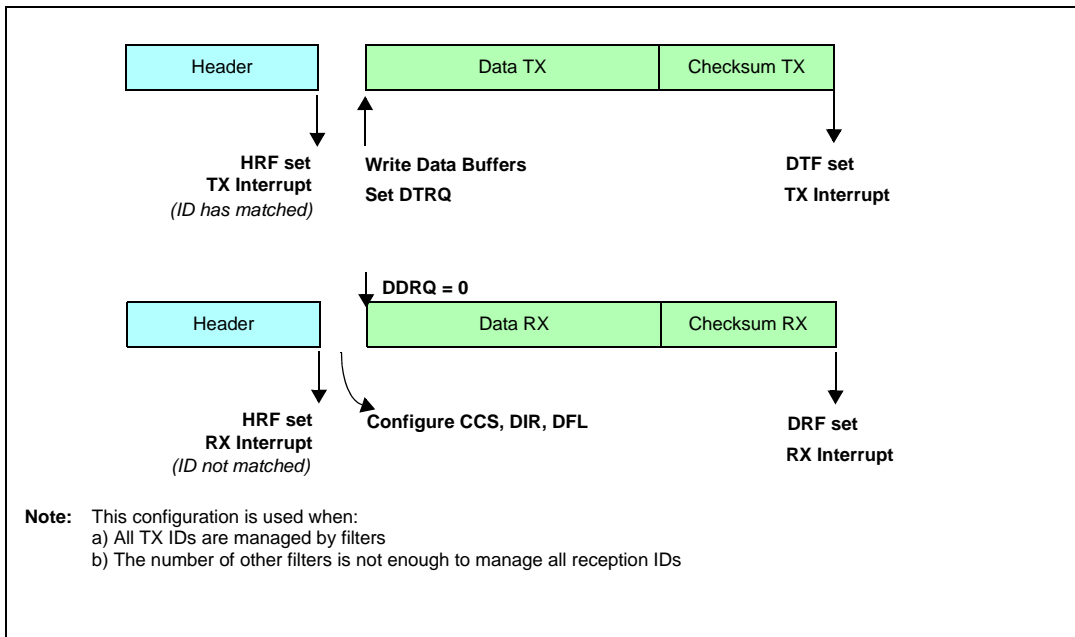


Figure 300. Programming consideration: slave node, TX filter, BF is set

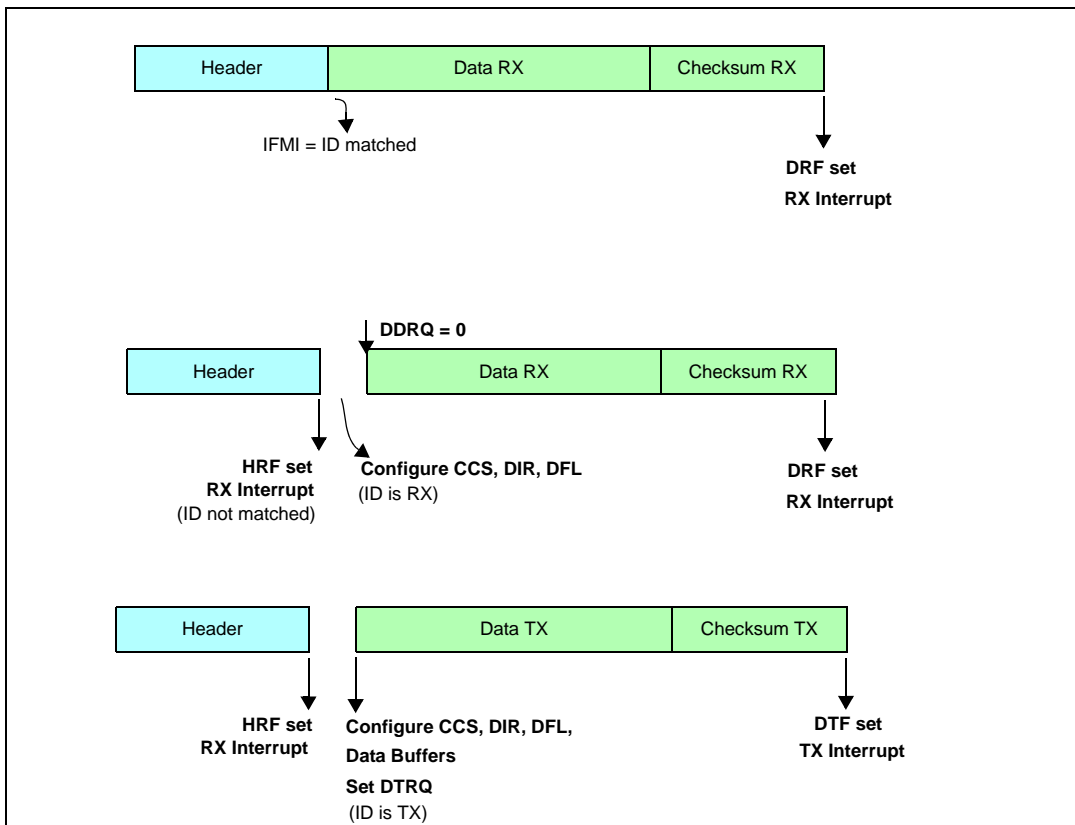


Figure 301. Programming consideration: slave node, RX filter, BF is set

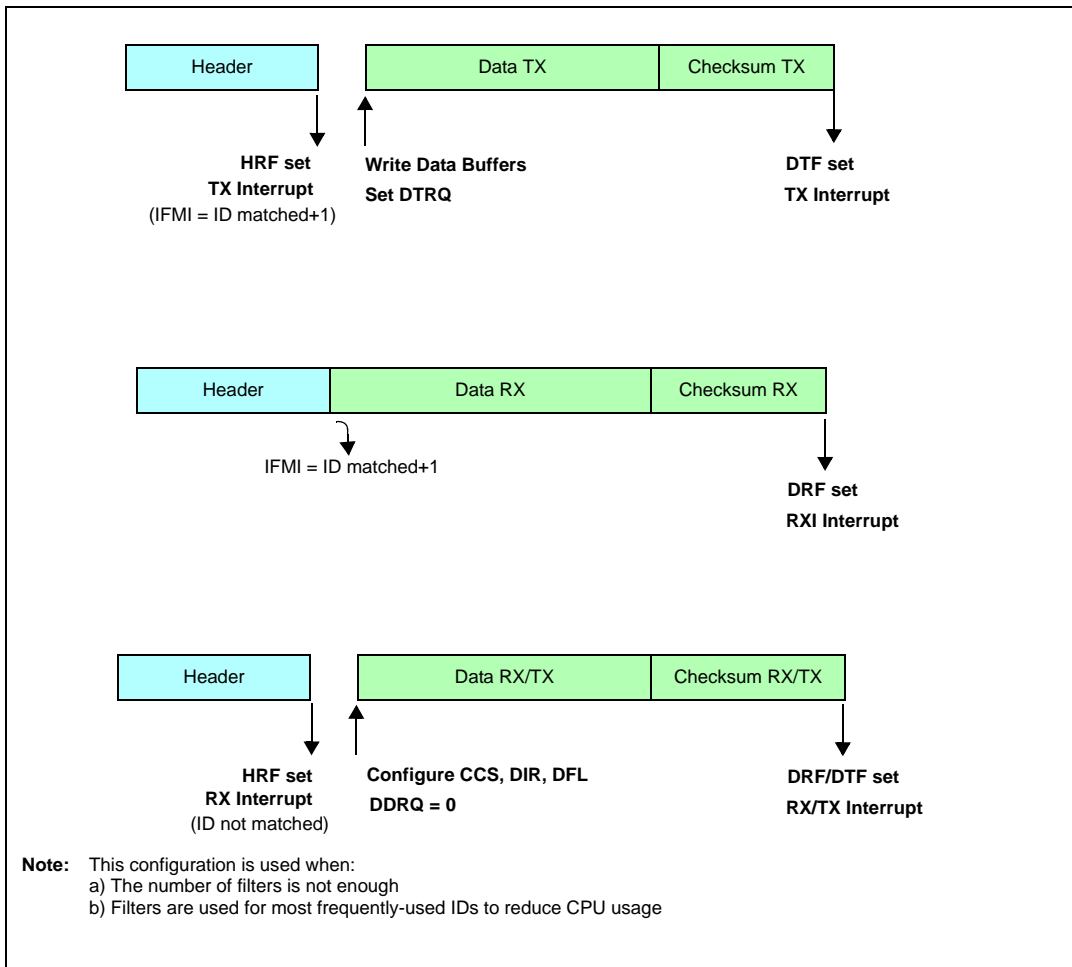


Figure 302. Programming consideration: slave node, TX filter, RX filter, BF is set

### 24.13.3 Extended frames

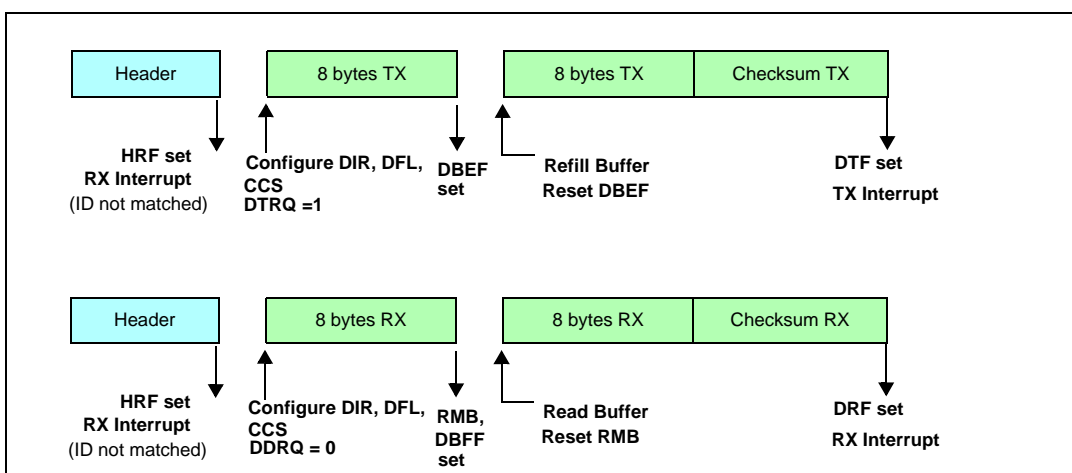


Figure 303. Programming consideration: extended frames

### 24.13.4 Timeout

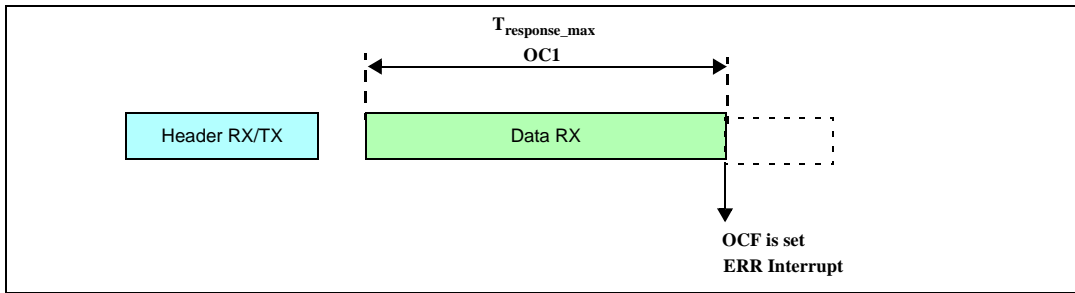


Figure 304. Programming consideration: response timeout

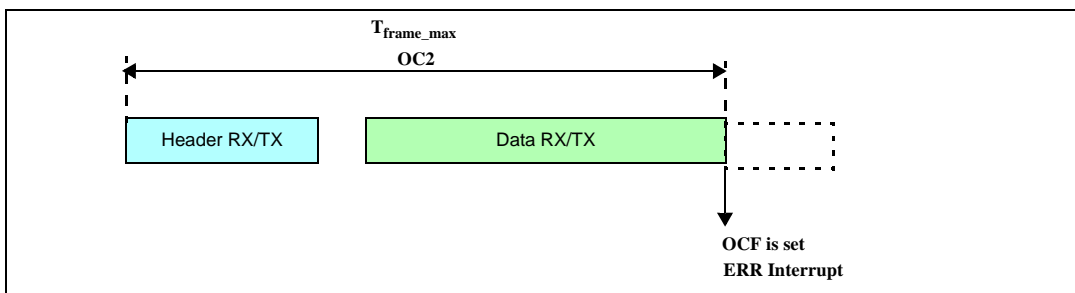


Figure 305. Programming consideration: frame timeout

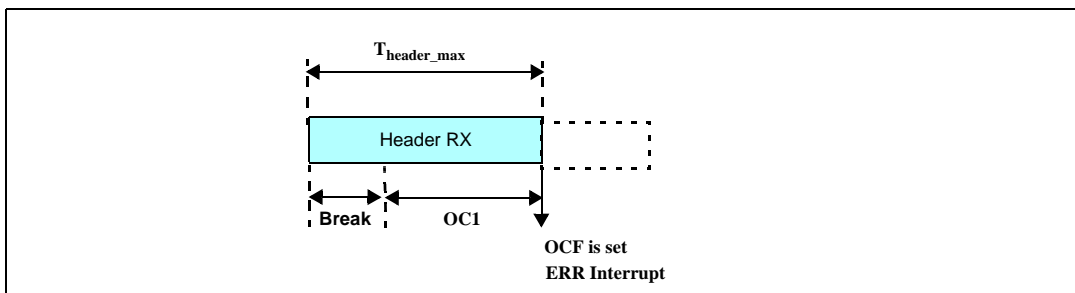


Figure 306. Programming consideration: header timeout

### 24.13.5 UART mode

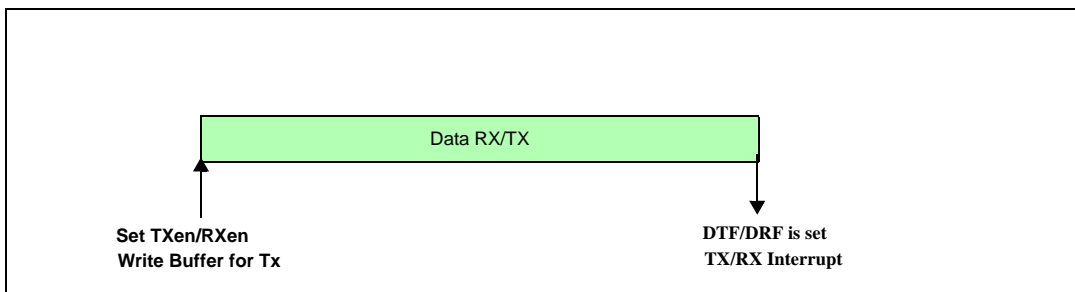


Figure 307. Programming consideration: UART mode

## 25 FlexCAN

### 25.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

#### 25.1.1 Device-specific features

The device has six Controller Area Network (FlexCAN) blocks.

- Each block supports 64 Message Buffers (MB).
- DMA support is not provided.
- It is possible to operate the bxcan FlexCAN bit timing logic with either system clock or 4–40 MHz fast external crystal oscillator clock (FXOSC).
- In the case of safe mode entry, the pad associated with CANTX can optionally be put into a high-impedance state (not recessive state)
- Modes of operation:
  - 4 functional modes: Normal (User and Supervisor), Freeze, Listen-Only and Loop-Back
  - 1 low-power mode (Disable mode)
- 1056 bytes (64 MBs) of RAM used for MB storage
- 256 bytes (64 MBs) of RAM used for individual Rx Mask Registers
- Hardware cancellation on Tx message buffers
- Module Configuration Register (MCR): Bits 5, 9, 12 and 13 are 'Reserved'
- Error and Status Register (ESR): Bit 31 is 'Reserved'

### 25.2 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification [Ref. 1]. A general block diagram is shown in [Figure 308](#), which describes the main sub-blocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask Registers. Support for up to 64 Message Buffers is provided. The functions of the sub-modules are described in subsequent sections.



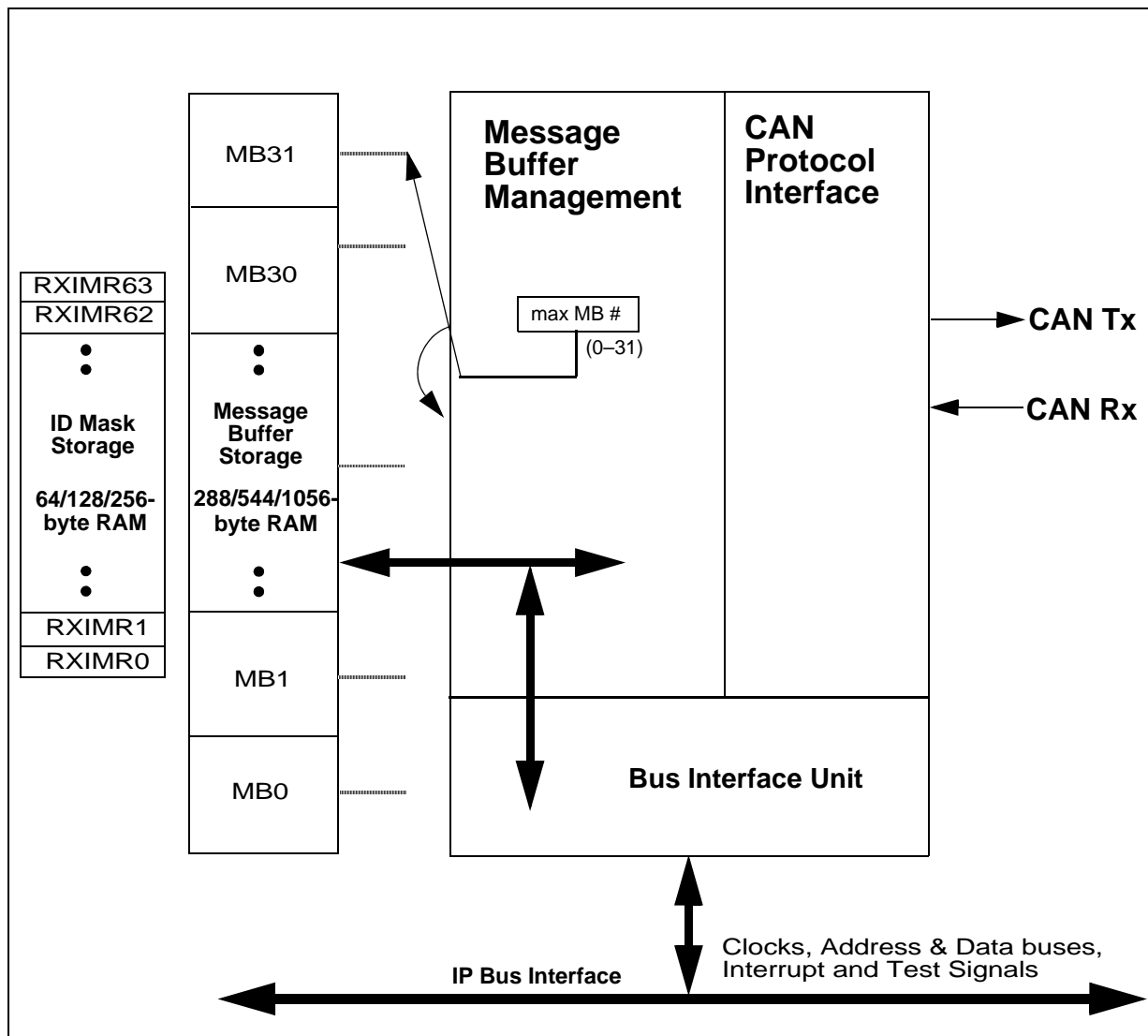


Figure 308. FlexCAN block diagram

### 25.2.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B [Ref. 1], which supports both standard and extended message frames. A flexible number of Message Buffers (16, 32 or 64) is also supported. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) sub-module handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) sub-module

controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the Bus Interface Unit.

## 25.2.2 FlexCAN module features

The FlexCAN module includes these distinctive features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbit/s
  - Content-related addressing
- Flexible Message Buffers (up to 64) of zero to eight bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes either 1056 bytes (64 MBs), 544 bytes (32 MBs) or 288 bytes (16 MBs) of RAM used for MB storage
- Includes either 256 bytes (64 MBs), 128 bytes (32 MBs) or 64 bytes (16 MBs) of RAM used for individual Rx Mask Registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask Register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes, with programmable wake up on bus activity

*Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported.*

### 25.2.3 Modes of operation

The FlexCAN module has four functional modes: Normal mode (User and Supervisor), Freeze mode, Listen-Only mode and Loop-Back mode. There is also a low-power mode (Disable mode).

- Normal mode (User or Supervisor):  
In Normal Mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor Modes differ in the access to some restricted control registers.
- Freeze mode:  
It is enabled when the FRZ bit in the MCR Register is asserted. If enabled, Freeze Mode is entered when the HALT bit in MCR is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See [Section 25.5.10.1: Freeze mode](#), for more information.
- Listen-Only mode:  
The module enters this mode when the LOM bit in the Control Register is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.
- Loop-Back mode:  
The module enters this mode when the LPB bit in the Control Register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic '1'). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.
- Module Disable mode:  
This low power mode is entered when the MCR[MDIS] bit is asserted by the CPU. When disabled, the module requests to disable the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. Exit from this mode is done by negating the MDIS bit in the MCR Register. See [Section 25.5.10.2: Module Disable mode](#), for more information.

## 25.3 External signal description

### 25.3.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in [Table 287](#) and described in more detail in the next subsections.

Table 287. FlexCAN Signals

Signal Name <sup>(1)</sup>	Direction	Description
CAN Rx	Input	CAN Receive Pin
CAN Tx	Output	CAN Transmit Pin

1. The actual MCU pins may have different names.

## 25.3.2 Signal descriptions

### 25.3.2.1 CAN Rx

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level '0'. Recessive state is represented by logic level '1'.

### 25.3.2.2 CAN Tx

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level '0'. Recessive state is represented by logic level '1'.

## 25.4 Memory map/register definition

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the MCU. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

### 25.4.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 64 MBs capability is shown in [Table 288](#).

All registers except for the MCR can be configured to have either supervisor or unrestricted access by programming the MCR[SUPV] bit.

The IFLAG2 and IMASK2 registers are considered reserved space when FlexCAN is configured with 16 or 32 MBs. The Rx Global Mask (RXGMASK), Rx Buffer 14 Mask (RX14MASK) and the Rx Buffer 15 Mask (RX15MASK) registers are provided for backwards compatibility, and are not used when the BCC bit in MCR is asserted.

The address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1056 and 256 bytes, respectively) only when FlexCAN is configured with 64 MBs. When it is configured with 16 MBs, the memory sizes are 288 and 64 bytes, so the address ranges 0x0180–0x047F and 0x08C0–0x097F are considered reserved space. When it is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if the BCC bit in MCR is negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

**Table 288. FlexCAN memory map**

Base addresses:		
0xFFFC_0000 (FlexCAN_0)		
0xFFFC_4000 (FlexCAN_1)		
0xFFFC_8000 (FlexCAN_2)		
0xFFFC_C000 (FlexCAN_3)		
0xFFFD_0000 (FlexCAN_4)		
0xFFFD_4000 (FlexCAN_5)		
Address offset	Register	Location
0x0000	Module Configuration (MCR)	<a href="#">on page 603</a>
0x0004	Control Register (CTRL)	<a href="#">on page 607</a>
0x0008	Free Running Timer (TIMER)	<a href="#">on page 610</a>
0x000C	Reserved	
0x0010	Rx Global Mask (RXGMASK)	<a href="#">on page 611</a>
0x0014	Rx Buffer 14 Mask (RX14MASK)	<a href="#">on page 613</a>
0x0018	Rx Buffer 15 Mask (RX15MASK)	<a href="#">on page 613</a>
0x001C	Error Counter Register (ECR)	<a href="#">on page 614</a>
0x0020	Error and Status Register (ESR)	<a href="#">on page 615</a>
0x0024	Interrupt Masks 2 (IMASK2)	<a href="#">on page 618</a>
0x0028	Interrupt Masks 1 (IMASK1)	<a href="#">on page 619</a>
0x002C	Interrupt Flags 2 (IFLAG2)	<a href="#">on page 620</a>
0x0030	Interrupt Flags 1 (IFLAG1)	<a href="#">on page 621</a>
0x0034–0x007F	Reserved	
0x0080–0x017F	Message Buffers MB0–MB15	—
0x0180–0x027F	Message Buffers MB16–MB31	—
0x0280–0x047F	Message Buffers MB32–MB63	—
0x0480–087F	Reserved	
0x0880–0x08BC	Rx Individual Mask Registers RXIMR0–RXIMR15	<a href="#">on page 622</a>
0x08C0–0x08FC	Rx Individual Mask Registers RXIMR16–RXIMR31	<a href="#">on page 622</a>
0x0900–0x097C	Rx Individual Mask Registers RXIMR32–RXIMR63	<a href="#">on page 622</a>

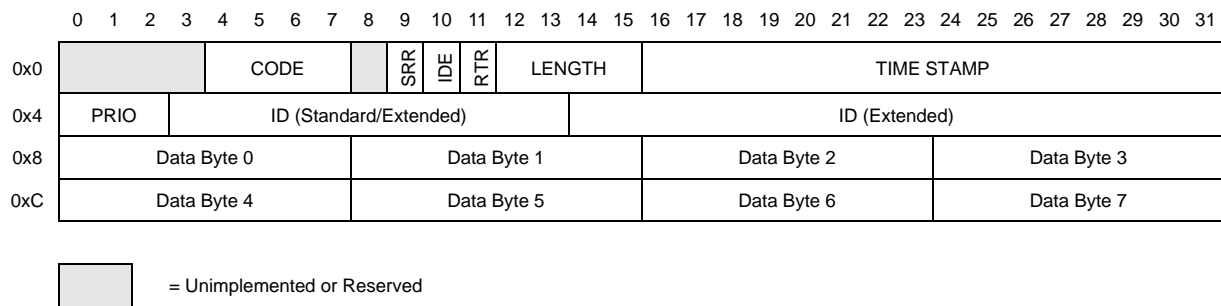
The FlexCAN module stores CAN messages for transmission and reception using a Message Buffer structure. Each individual MB is formed by 16 bytes mapped on memory as described in [Table 289](#). [Table 289](#) shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

**Table 289. Message Buffer MB0 memory mapping**

Address Offset	MB Field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

### 25.4.2 Message Buffer Structure

The Message Buffer structure used by the FlexCAN module is represented in [Figure 309](#). Both Extended and Standard Frames (29-bit Identifier and 11-bit Identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.



**Figure 309. Message Buffer Structure**

**Table 290. Message Buffer Structure field description**

Field	Description
CODE	<b>Message Buffer Code</b> This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding is shown in <a href="#">Table 291</a> and <a href="#">Table 292</a> . See <a href="#">Section 25.5: Functional description</a> , for additional information.
SRR	<b>Substitute Remote Request</b> Fixed recessive bit, used only in extended format. It must be set to '1' by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 1 = Recessive value is compulsory for transmission in Extended Format frames 0 = Dominant is not a valid value for transmission in Extended Format frames
IDE	<b>ID Extended Bit</b> This bit identifies whether the frame format is standard or extended. 1 = Frame format is extended 0 = Frame format is standard

**Table 290. Message Buffer Structure field description (continued)**

Field	Description
RTR	<p><b>Remote Transmission Request</b></p> <p>This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as '1' (recessive) and receives it as '0' (dominant), it is interpreted as arbitration loss. If this bit is transmitted as '0' (dominant), then if it is received as '1' (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission.</p> <p>1 = Indicates the current MB has a Remote Frame to be transmitted                      0 = Indicates the current MB has a Data Frame to be transmitted</p> <p>Note: Do not configure the last Message Buffer to be the RTR frame.</p>
LENGTH	<p><b>Length of Data in Bytes</b></p> <p>This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see <a href="#">Figure 309</a>). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field.</p>
TIME STAMP	<p><b>Free-Running Counter Time Stamp</b></p> <p>This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.</p>
PRIO	<p><b>Local priority</b></p> <p>This 3-bit field is only used when LPRIO_EN bit is set in MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See <a href="#">Section 25.5.4: Arbitration process</a>.</p>
ID	<p><b>Frame Identifier</b></p> <p>In Standard Frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.</p>
DATA	<p><b>Data Field</b></p> <p>Up to eight bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.</p>

**Table 291. Message Buffer Code for Rx buffers**

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: MB is not active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.

**Table 291. Message Buffer Code for Rx buffers**

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. See <a href="#">Section 25.5.6: Matching process</a> , for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. See <a href="#">Section 25.5.6: Matching process</a> , for details about overrun behavior.
0XY1 <sup>(1)</sup>	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1. Note that for Tx MBs (see [Table 292](#)), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR register.

**Table 292. Message Buffer Code for Tx buffers**

RTR	Initial Tx code	Code after successful transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.

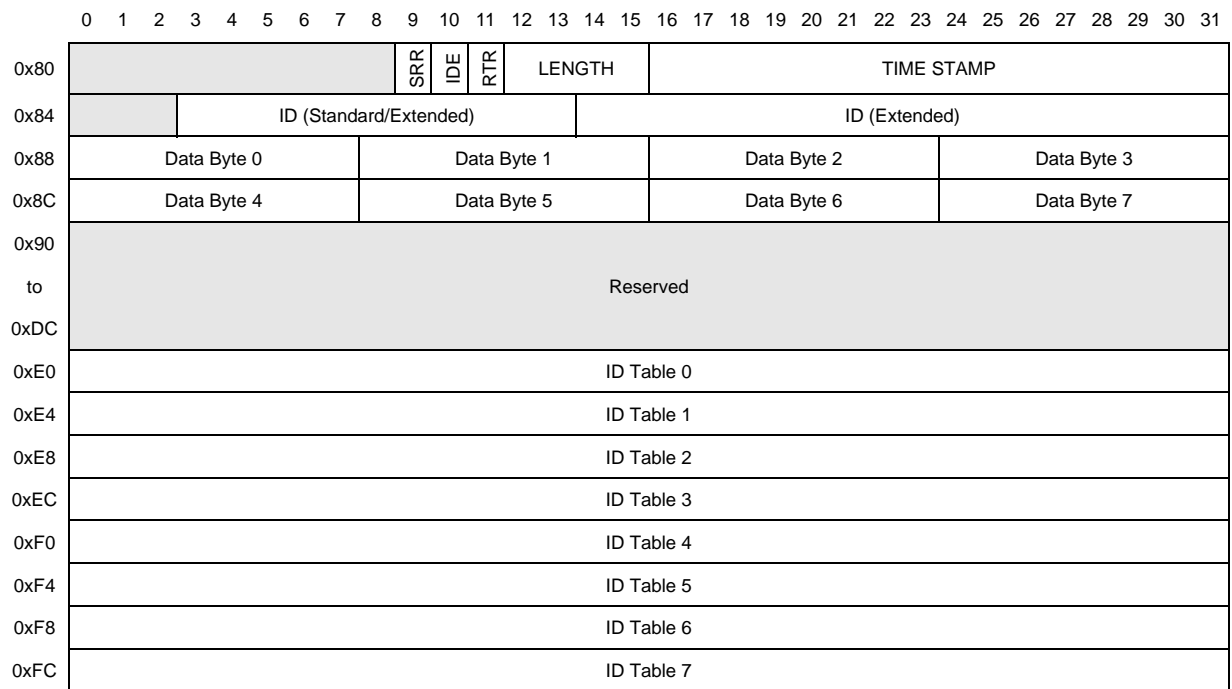


Table 292. Message Buffer Code for Tx buffers (continued)

RTR	Initial Tx code	Code after successful transmission	Description
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to '1110' to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to '1010' to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to '1010'. The CPU can also write this code with the same effect.

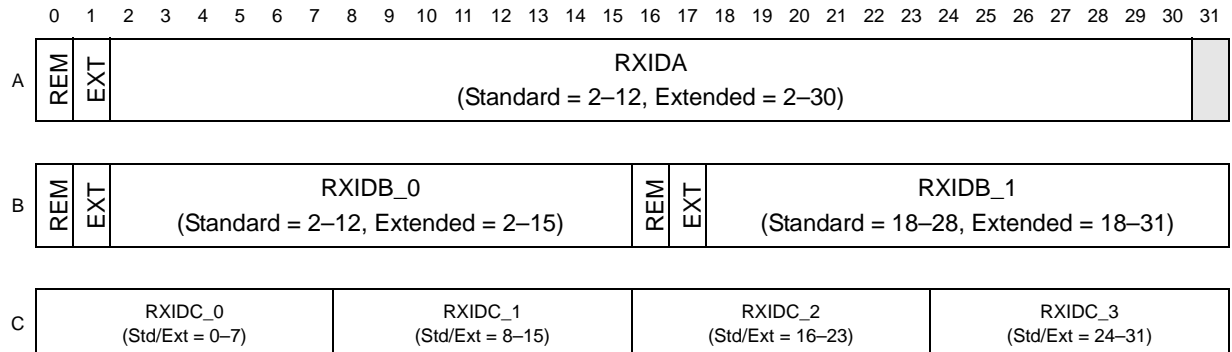
### 25.4.3 Rx FIFO structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. [Figure 310](#) shows the Rx FIFO data structure. The region 0x80-0x8C contains an MB structure which is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDC is reserved for internal use of the FIFO engine. The region 0xE0-0xFC contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. [Figure 311](#) shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the MCR. Note that all elements of the table must have the same format. See [Section 25.5.8: Rx FIFO](#), for more information.



= Unimplemented or Reserved

**Figure 310. Rx FIFO structure**



= Unimplemented or Reserved

**Figure 311. ID Table 0–7**

**Table 293. Rx FIFO Structure field description**

Field	Description
REM	<b>Remote Frame</b> This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 1 = Remote Frames can be accepted and data frames are rejected 0 = Remote Frames are rejected and data frames can be accepted
EXT	<b>Extended Frame</b> Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 1 = Extended frames can be accepted and standard frames are rejected 0 = Extended frames are rejected and standard frames can be accepted
RXIDA	<b>Rx Frame Identifier (Format A)</b> Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0, RXIDB_1	<b>Rx Frame Identifier (Format B)</b> Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0, RXIDC_1, RXIDC_2, RXIDC_3	<b>Rx Frame Identifier (Format C)</b> Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

### 25.4.4 Register descriptions

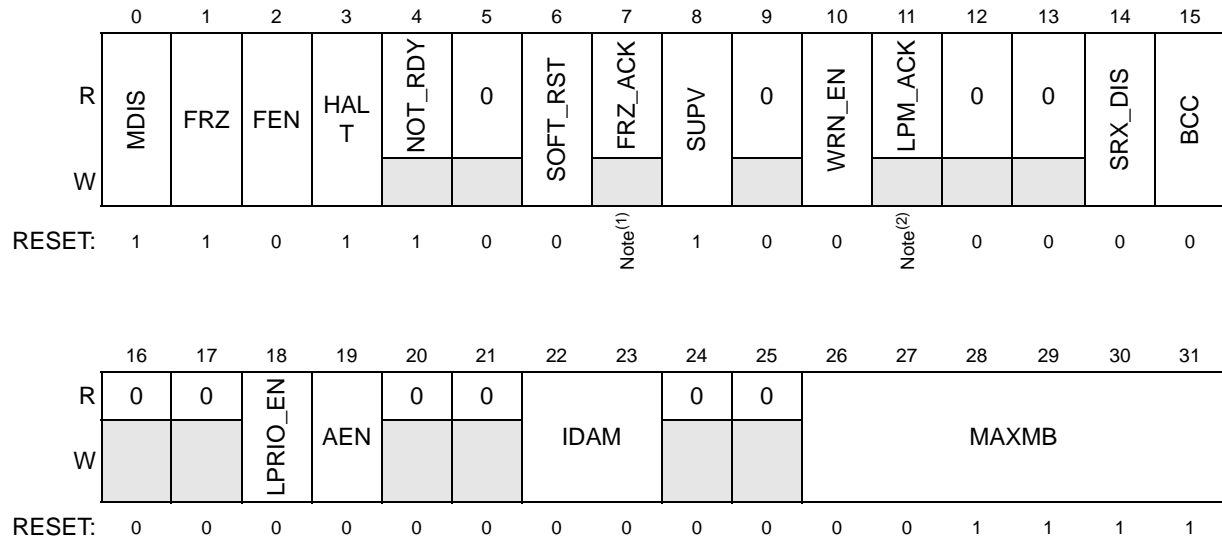
The FlexCAN registers are described in this section in ascending address order.

#### 25.4.4.1 Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. This register can be accessed at any time, however some fields must be changed only during Freeze Mode. Find more information in the fields descriptions ahead.

Offset: 0x0000

Access: Supervisor read/write



**Figure 312. Module Configuration Register (MCR)**

1. Different on various platforms, but it is always the opposite of the MDIS reset value.
2. Different on various platforms, but it is always the same as the MDIS reset value.

**Table 294. MCR field descriptions**

Field	Description
MDIS	<p>Module Disable</p> <p>This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. This is the only bit in MCR not affected by soft reset. See <a href="#">Section 25.5.10.2: Module Disable mode</a>, for more information.</p> <p>1 = Disable the FlexCAN module 0 = Enable the FlexCAN module</p>
FRZ	<p>Freeze Enable</p> <p>The FRZ bit specifies the FlexCAN behavior when the HALT bit in the MCR Register is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode.</p> <p>1 = Enabled to enter Freeze Mode 0 = Not enabled to enter Freeze Mode</p>
FEN	<p>FIFO Enable</p> <p>This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80-0xFF) is used by the FIFO engine. See <a href="#">Section 25.4.3: Rx FIFO structure</a>, and <a href="#">Section 25.5.8: Rx FIFO</a>, for more information. This bit must be written in Freeze mode only.</p> <p>1 = FIFO enabled 0 = FIFO not enabled</p>

**Table 294. MCR field descriptions (continued)**

Field	Description
<p>HALT</p>	<p>Halt FlexCAN</p> <p>Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the Message Buffers and Control Register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See <a href="#">Section 25.5.10.1: Freeze mode</a>, for more information.</p> <p>1 = Enters Freeze Mode if the FRZ bit is asserted. 0 = No Freeze Mode request.</p>
<p>NOT_RDY</p>	<p>FlexCAN Not Ready</p> <p>This read-only bit indicates that FlexCAN is in Disable Mode or Freeze Mode. It is negated once FlexCAN has exited these modes.</p> <p>1 = FlexCAN module is in Disable Mode or Freeze Mode 0 = FlexCAN module is in Normal Mode, Listen-Only Mode or Loop-Back Mode</p>
<p>SOFT_RST</p>	<p>Soft Reset</p> <p>When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IMASK2, IFLAG1, IFLAG2. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected:</p> <p>CTRL RXIMR0–RXIMR63 RXGMASK, RX14MASK, RX15MASK all Message Buffers</p> <p>The SOFT_RST bit can be asserted directly by the CPU when it writes to the MCR Register, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed.</p> <p>Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied.</p> <p>1 = Resets the registers marked as “affected by soft reset” in <a href="#">Table 288</a> 0 = No reset request</p>
<p>FRZ_ACK</p>	<p>Freeze Mode Acknowledge</p> <p>This read-only bit indicates that FlexCAN is in Freeze Mode and its prescaler is stopped. The Freeze Mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered Freeze Mode. If Freeze Mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze Mode is requested while FlexCAN is in any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See <a href="#">Section 25.5.10.1: Freeze mode</a>, for more information.</p> <p>1 = FlexCAN in Freeze Mode, prescaler stopped 0 = FlexCAN not in Freeze Mode, prescaler running</p>

Table 294. MCR field descriptions (continued)

Field	Description
SUPV	<p>Supervisor Mode</p> <p>This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of <a href="#">Table 288</a>. Reset value of this bit is '1', so the affected registers start with Supervisor access restrictions. This bit should be written in Freeze mode only.</p> <p>1 = Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location</p> <p>0 = Affected registers are in Unrestricted memory space</p>
WRN_EN	<p>Warning Interrupt Enable</p> <p>When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. This bit must be written in Freeze mode only.</p> <p>1 = TWRN_INT and RWRN_INT bits are set when the respective error counter transition from &lt;96 to ≥ 96.</p> <p>0 = TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters.</p>
LPM_ACK	<p>Low Power Mode Acknowledge</p> <p>This read-only bit indicates that FlexCAN is in Disable Mode. This mode cannot be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually entered low power mode. See <a href="#">Section 25.5.10.2: Module Disable mode</a>, for more information.</p> <p>1 = FlexCAN is in Disable Mode</p> <p>0 = FlexCAN not in any low-power mode</p>
SRX_DIS	<p>Self Reception Disable</p> <p>This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. This bit must be written in Freeze mode only.</p> <p>1 = Self reception disabled</p> <p>0 = Self reception enabled</p>
BCC	<p>Backwards Compatibility Configuration</p> <p>This bit is provided to support Backwards Compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied:</p> <p>For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK and RX15MASK.</p> <p>The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to '0110' (overrun).</p> <p>Upon reset this bit is negated, allowing legacy software to work without modification. This bit must be written in Freeze mode only.</p> <p>1 = Individual Rx masking and queue feature are enabled.</p> <p>0 = Individual Rx masking and queue feature are disabled.</p>

**Table 294. MCR field descriptions (continued)**

Field	Description
LPRIO_EN	<p>Local Priority Enable</p> <p>This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It is used to extend the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. This bit must be written in Freeze mode only.</p> <p>1 = Local Priority enabled 0 = Local Priority disabled</p>
AEN	<p>Abort Enable</p> <p>This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. This bit must be written in Freeze mode only.</p> <p>1 = Abort enabled 0 = Abort disabled</p>
IDAM	<p>ID Acceptance Mode</p> <p>This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in <a href="#">Table 295</a>. Note that all elements of the table are configured at the same time by this field (they are all the same format). See <a href="#">Section 25.4.3: Rx FIFO structure</a>. This bit must be written in Freeze mode only.</p>
MAXMB	<p>Maximum Number of Message Buffers</p> <p>This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field must be changed only while the module is in Freeze Mode.</p> <p>Maximum MBs in use = MAXMB + 1.</p> <p><b>Note:</b> MAXMB must be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN can transmit and receive wrong messages.</p>

**Table 295. IDAM coding**

IDAM	Format	Explanation
0b00	A	One full ID (standard or extended) per filter element.
0b01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
0b10	C	Four partial 8-bit IDs (standard or extended) per filter element.
0b11	D	All frames rejected.

**25.4.4.2 Control Register (CTRL)**

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF\_MSK, ERR\_MSK, TWRN\_MSK, RWRN\_MSK and BOFF\_REC bits, that can be accessed at any time.

Offset: 0x0004

Access: Read/write

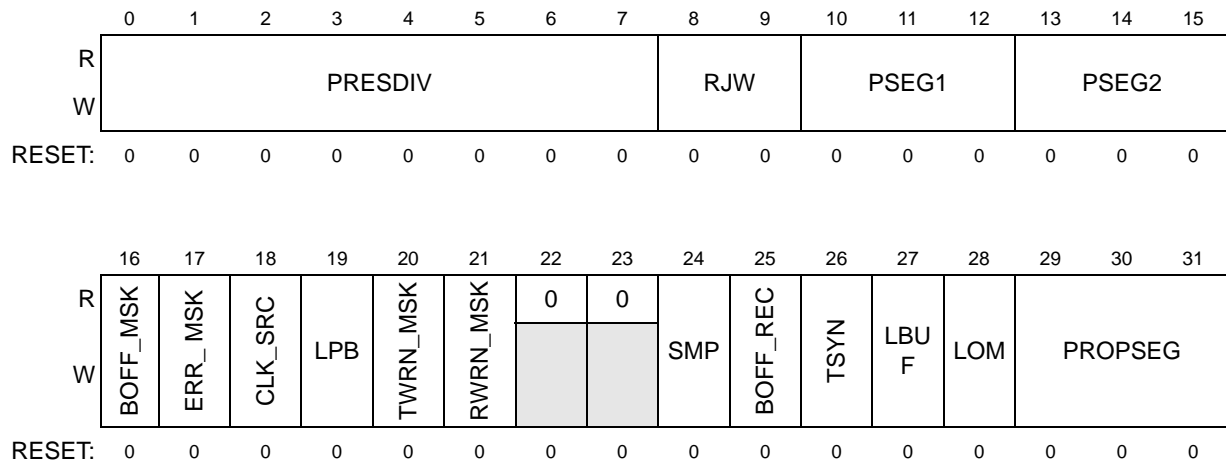


Figure 313. Control Register (CTRL)

Table 296. CTRL field descriptions

Field	Description
PRESDIV	<p>Prescaler Division Factor</p> <p>This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to <a href="#">Section 25.5.9.4: Protocol Timing</a>. This bit must be written in Freeze mode only.</p> <p><math>Sclock\ frequency = CPI\ clock\ frequency / (PRESDIV + 1)</math></p>
RJW	<p>Resync Jump Width</p> <p>This 2-bit field defines the maximum number of time quanta<sup>(1)</sup> that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3. This bit must be written in Freeze mode only.</p> <p><math>Resync\ Jump\ Width = RJW + 1.</math></p>
PSEG1	<p>Phase Segment 1</p> <p>This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. This bit must be written in Freeze mode only.</p> <p><math>Phase\ Buffer\ Segment\ 1 = (PSEG1 + 1) \times Time-Quanta.</math></p>
PSEG2	<p>Phase Segment 2</p> <p>This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. This bit must be written in Freeze mode only.</p> <p><math>Phase\ Buffer\ Segment\ 2 = (PSEG2 + 1) \times Time-Quanta.</math></p>
BOFF_MSK	<p>Bus Off Mask</p> <p>This bit provides a mask for the Bus Off Interrupt.</p> <p>1 = Bus Off interrupt enabled 0 = Bus Off interrupt disabled</p>



**Table 296. CTRL field descriptions (continued)**

Field	Description
ERR_MSK	<p>Error Mask</p> <p>This bit provides a mask for the Error Interrupt.</p> <p>1 = Error interrupt enabled</p> <p>0 = Error interrupt disabled</p>
CLK_SRC	<p>CAN Engine Clock Source</p> <p>This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit must only be changed while the module is in Disable Mode. See <a href="#">Section 25.5.9.4: Protocol Timing</a>, for more information.</p> <p>1 = The CAN engine clock source is the bus clock</p> <p>0 = The CAN engine clock source is the oscillator clock</p>
TWRN_MSK	<p>Tx Warning Interrupt Mask</p> <p>This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated.</p> <p>1 = Tx Warning Interrupt enabled</p> <p>0 = Tx Warning Interrupt disabled</p>
RWRN_MSK	<p>Rx Warning Interrupt Mask</p> <p>This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated.</p> <p>1 = Rx Warning Interrupt enabled</p> <p>0 = Rx Warning Interrupt disabled</p>
LPB	<p>Loop Back</p> <p>This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic '1'). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. This bit must be written in Freeze mode only.</p> <p>1 = Loop Back enabled</p> <p>0 = Loop Back disabled</p>
SMP	<p>Sampling Mode</p> <p>This bit defines the sampling mode of CAN bits at the Rx input. This bit must be written in Freeze mode only.</p> <p>1 = Three samples are used to determine the value of the received bit: the regular one (sample point) and 2 preceding samples, a majority rule is used</p> <p>0 = Just one sample is used to determine the bit value</p>

Table 296. CTRL field descriptions (continued)

Field	Description
BOFF_REC	<p>Bus Off Recovery Mode</p> <p>This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will re-synchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was negated when the module entered Bus Off, asserting it during Bus Off will not be effective for the current Bus Off recovery.</p> <p>1 = Automatic recovering from Bus Off state disabled 0 = Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B</p>
TSYN	<p>Timer Sync Mode</p> <p>This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special "SYNC" message (i.e., global network time). If the FEN bit in MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. This bit must be written in Freeze mode only.</p> <p>1 = Timer Sync feature enabled 0 = Timer Sync feature disabled</p>
LBUF	<p>Lowest Buffer Transmitted First</p> <p>This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the LPRIO_EN bit does not affect the priority arbitration. This bit must be written in Freeze mode only.</p> <p>1 = Lowest number buffer is transmitted first 0 = Buffer with highest priority is transmitted first</p>
LOM	<p>Listen-Only Mode</p> <p>This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. This bit must be written in Freeze mode only.</p> <p>1 = FlexCAN module operates in Listen Only Mode 0 = Listen Only Mode is deactivated</p>
PROPSEG	<p>Propagation Segment</p> <p>This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. This bit must be written in Freeze mode only.</p> <p>Propagation Segment Time = (PROPSEG + 1) * Time-Quanta. Time-Quantum = one Sclock period.</p>

1. One time quantum is equal to the Sclock period.

#### 25.4.4.3 Free Running Timer (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is

received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Offset: 0x0008

Access: Read/write

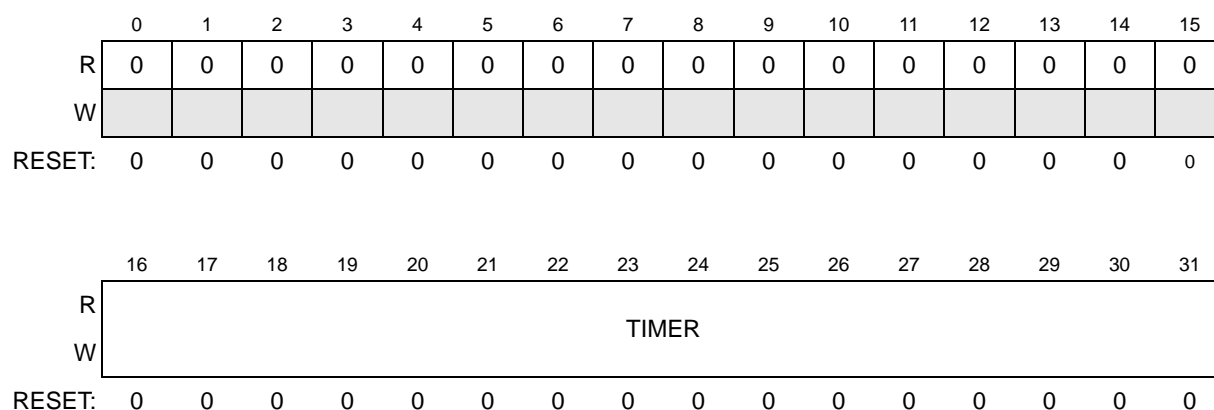


Figure 314. Free Running Timer (TIMER)

Table 297. TIMER field descriptions

Field	Description
TIMER	Free-running timer counter. The timer starts from 0x0000 after reset, counts linearly to 0xFFFF, and wraps around.

#### 25.4.4.4 Rx Global Mask (RXGMASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. Setting the BCC bit in MCR causes the RXGMASK Register to have no effect on the module operation. This register is always effective.

RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in MCR is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

See [Section 25.5.8: Rx FIFO](#), for important details on usage of RXGMASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

During CAN messages reception by FlexCAN, the RXGMASK (Rx Global Mask) is used as acceptance mask for most of the Rx Message Buffers (MB). When the FIFO Enable bit in the FlexCAN Module Configuration Register (CANx\_MCR[FEN], bit 2) is set, the RXGMASK also applies to most of the elements of the ID filter table. However there is a misalignment between the position of the ID field in the Rx MB and in RXIDA, RXIDB and RXIDC fields of the ID Tables. In fact RXIDA filter in the ID Tables is shifted one bit to the left from Rx MBs ID position as shown below:

- Rx MB ID = bits 3–31 of ID word corresponding to message ID bits 0–28
- RXIDA = bits 2–30 of ID Table corresponding to message ID bits 0–28

The mask bits one-to-one correspondence occurs with the filters bits, not with the incoming message ID bits. This leads the RXGMASK to affect Rx MB and Rx FIFO filtering in different ways.

For example, if the user intends to mask out the bit 24 of the ID filter of Message Buffers then the RXGMASK will be configured as 0xffff\_ffef. As result, bit 24 of the ID field of the incoming message will be ignored during filtering process for Message Buffers. This very same configuration of RXGMASK would lead bit 24 of RXIDA to be "don't care" and thus bit 25 of the ID field of the incoming message would be ignored during filtering process for Rx FIFO.

Similarly, both RXIDB and RXIDC filters have multiple misalignments with regards to position of ID field in Rx MBs, which can lead to erroneous masking during filtering process for either Rx FIFO or MBs.

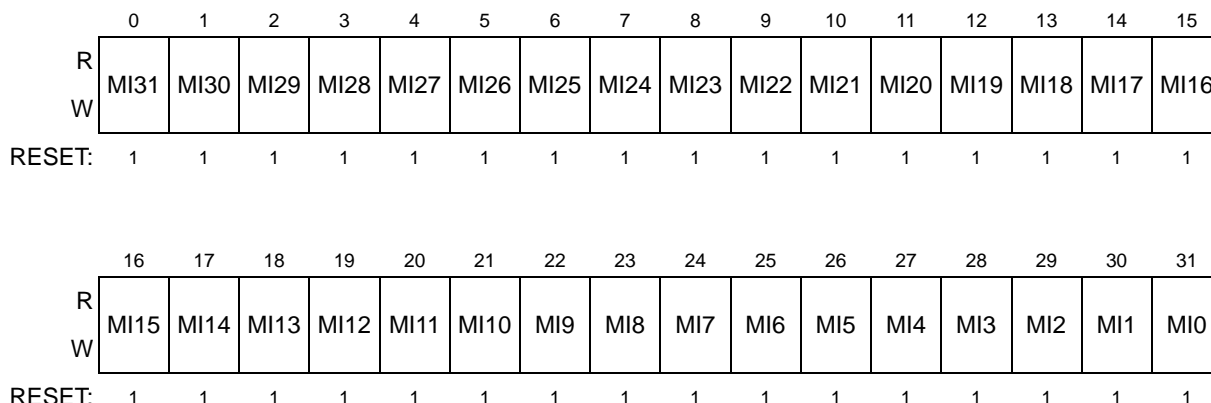
RX14MASK (Rx 14 Mask) and RX15MASK (Rx 15 Mask) have the same structure as the RXGMASK. This includes the misalignment problem between the position of the ID field in the Rx MBs and in RXIDA, RXIDB and RXIDC fields of the ID Tables.

Therefore it is recommended that one of the following actions be taken to avoid problems:

- Do not enable the Rx FIFO. If CANx\_MCR[FEN]=0 then the Rx FIFO is disabled and thus the masks RXGMASK, RX14MASK and RX15MASK do not affect it.
- Enable Rx Individual Mask Registers. If the Backwards Compatibility Configuration bit in the FlexCAN Module Configuration Register (CANx\_MCR[BCC], bit 15) is set then the Rx Individual Mask Registers (RXIMR0–63) are enabled and thus the masks RXGMASK, RX14MASK and RX15MASK are not used.
- Do not use masks RXGMASK, RX14MASK and RX15MASK (i.e., let them in reset value which is 0xffff\_ffff) when CANx\_MCR[FEN]=1 and CANx\_MCR[BCC]=0. In this case, filtering processes for both Rx MBs and Rx FIFO are not affected by those masks.
- Do not configure any MB as Rx (i.e., let all MBs as either Tx or inactive) when CANx\_MCR[FEN]=1 and CANx\_MCR[BCC]=0. In this case, the masks RXGMASK, RX14MASK and RX15MASK can be used to affect ID Tables without affecting filtering process for Rx MBs.

Offset: 0x0010

Access: Read/write



**Figure 315. Rx Global Mask Register (RXGMASK)**

**Table 298. RXGMASK field descriptions**

Field	Description
MI <sub>n</sub>	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1 = The corresponding bit in the filter is checked against the one received 0 = The corresponding bit in the filter is “don’t care”

**25.4.4.5 Rx 14 Mask (RX14MASK)**

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. Setting the BCC bit in MCR causes the RX14MASK Register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in MCR is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

See [Section 25.5.8: Rx FIFO](#), for important details on usage of RX14MASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x14
- Reset Value: 0xFFFF\_FFFF

**25.4.4.6 Rx 15 Mask (RX15MASK)**

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. Setting the BCC bit in MCR causes the RX15MASK Register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in MCR is set (FIFO enabled), the RXG15MASK also

applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

Refer to [Section 25.5.8: Rx FIFO](#), for important details on usage of RX15MASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x18
- Reset Value: 0xFFFF\_FFFF

#### 25.4.4.7 Error Counter Register (ECR)

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: Transmit Error Counter (TX\_ERR\_COUNTER field) and Receive Error Counter (RX\_ERR\_COUNTER field). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except in Freeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register while in Freeze Mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g., transmit 'Error Active' or 'Error Passive' flag, delay its transmission start time ('Error Passive') and avoid any influence on the bus when in 'Bus Off' state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of TX\_ERR\_COUNTER or RX\_ERR\_COUNTER increases to be greater than or equal to 128, the FLT\_CONF field in the Error and Status Register is updated to reflect 'Error Passive' state.
- If the FlexCAN state is 'Error Passive', and either TX\_ERR\_COUNTER or RX\_ERR\_COUNTER decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT\_CONF field in the Error and Status Register is updated to reflect 'Error Active' state.
- If the value of TX\_ERR\_COUNTER increases to be greater than 255, the FLT\_CONF field in the Error and Status Register is updated to reflect 'Bus Off' state, and an interrupt may be issued. The value of TX\_ERR\_COUNTER is then reset to zero.
- If FlexCAN is in 'Bus Off' state, then Tx\_Err\_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TX\_ERR\_COUNTER is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TX\_ERR\_COUNTER. When TX\_ERR\_COUNTER reaches the value of 128, the FLT\_CONF field in the Error and Status Register is updated to be 'Error Active' and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TX\_ERR\_COUNTER value.
- If during system start-up, only one node is operating, then its TX\_ERR\_COUNTER increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK\_ERR bit in the Error and Status Register). After the transition to

‘Error Passive’ state, the TX\_ERR\_COUNTER does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.

- If the RX\_ERR\_COUNTER increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘Error Active’ state.

Offset: 0x001C

Access: Read/write

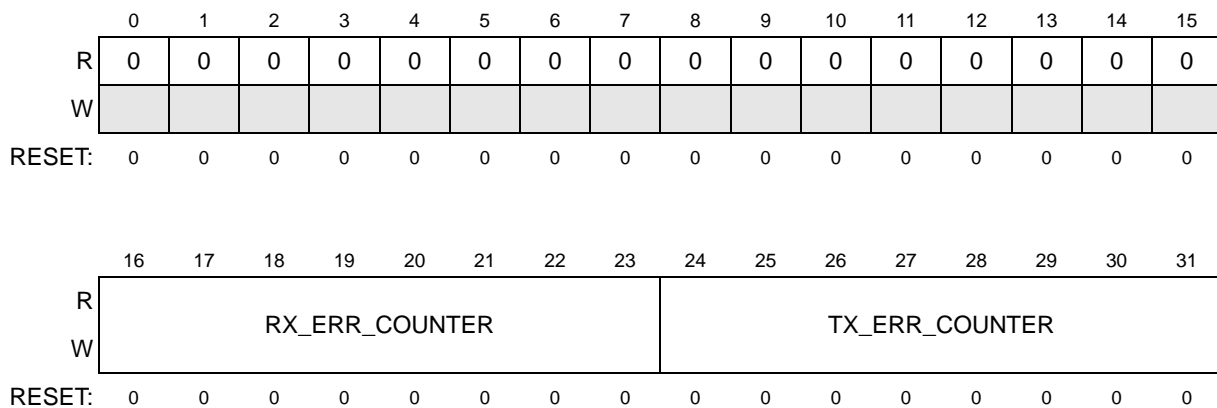


Figure 316. Error Counter Register (ECR)

Table 299. ECR field descriptions

Field	Description
RX_ERROR_COUNTER	Receive Error Counter. See the text of this section for a detailed description of this field and how it interacts with TX_ERROR_COUNTER.
TX_ERROR_COUNTER	Transmit Error Counter. See the text of this section for a detailed description of this field and how it interacts with RX_ERROR_COUNTER.

25.4.4.8 Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device and it is the source of four interrupts to the CPU. The reported error conditions (bits 16–21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16–23. Bits 22–28 are status bits.

Most bits in this register are read only, except TWRN\_INT, RWRN\_INT, BOFF\_INT, and ERR\_INT, that are interrupt flags that can be cleared by writing ‘1’ to them (writing ‘0’ has no effect). See [Section 25.5.11: Interrupts](#), for more details.

Offset: 0x0020

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF	0	BOFF_INT	ERR_INT	0	
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 317. Error and Status Register (ESR)

Table 300. ESR field descriptions

Field	Description
TWRN_INT	<p>TWRN_INT — Tx Warning Interrupt Flag</p> <p>If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from '0' to '1', meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to '1'. Writing '0' has no effect.</p> <p>1 = The Tx error counter transition from &lt; 96 to ≥ 96                      0 = No such occurrence</p>
RWRN_INT	<p>RWRN_INT — Rx Warning Interrupt Flag</p> <p>If the WRN_EN bit in MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from '0' to '1', meaning that the Rx error counters reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to '1'. Writing '0' has no effect.</p> <p>1 = The Rx error counter transition from &lt; 96 to ≥ 96                      0 = No such occurrence</p>
BIT1_ERR	<p>BIT1_ERR — Bit1 Error</p> <p>This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message.</p> <p>1 = At least one bit sent as recessive is received as dominant                      0 = No such occurrence</p> <p><b>Note:</b> This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.</p>



Table 300. ESR field descriptions (continued)

Field	Description
BIT0_ERR	<p>BIT0_ERR — Bit0 Error</p> <p>This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message.</p> <p>1 = At least one bit sent as dominant is received as recessive 0 = No such occurrence</p>
ACK_ERR	<p>ACK_ERR — Acknowledge Error</p> <p>This bit indicates that an Acknowledge Error has been detected by the transmitter node, i.e., a dominant bit has not been detected during the ACK SLOT.</p> <p>1 = An ACK error occurred since last read of this register 0 = No such occurrence</p>
CRC_ERR	<p>CRC_ERR — Cyclic Redundancy Check Error</p> <p>This bit indicates that a CRC Error has been detected by the receiver node, i.e., the calculated CRC is different from the received.</p> <p>1 = A CRC error occurred since last read of this register. 0 = No such occurrence</p>
FRM_ERR	<p>FRM_ERR — Form Error</p> <p>This bit indicates that a Form Error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit.</p> <p>1 = A Form Error occurred since last read of this register 0 = No such occurrence</p>
STF_ERR	<p>STF_ERR — Stuffing Error</p> <p>This bit indicates that a Stuffing Error has been detected.</p> <p>1 = A Stuffing Error occurred since last read of this register. 0 = No such occurrence.</p>
TX_WRN	<p>TX Error Warning</p> <p>This bit indicates when repetitive errors are occurring during message transmission.</p> <p>1 = TX_Err_Counter <math>\geq</math> 96 0 = No such occurrence</p>
RX_WRN	<p>Rx Error Counter</p> <p>This bit indicates when repetitive errors are occurring during message reception.</p> <p>1 = Rx_Err_Counter <math>\geq</math> 96 0 = No such occurrence</p>
IDLE	<p>CAN bus IDLE state</p> <p>This bit indicates when CAN bus is in IDLE state.</p> <p>1 = CAN bus is now IDLE 0 = No such occurrence</p>
TXRX	<p>Current FlexCAN status (transmitting/receiving)</p> <p>This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted.</p> <p>1 = FlexCAN is transmitting a message (IDLE=0) 0 = FlexCAN is receiving a message (IDLE=0)</p>

**Table 300. ESR field descriptions (continued)**

Field	Description
FLT_CONF	<p>Fault Confinement State</p> <p>This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in <a href="#">Table 301</a>. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted.</p>
BOFF_INT	<p>‘Bus Off’ Interrupt</p> <p>This bit is set when FlexCAN enters ‘Bus Off’ state. If the corresponding mask bit in the Control Register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect.</p> <p>1 = FlexCAN module entered ‘Bus Off’ state 0 = No such occurrence</p>
ERR_INT	<p>Error Interrupt</p> <p>This bit indicates that at least one of the Error Bits (bits 16-21) is set. If the corresponding mask bit in the Control Register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect.</p> <p>1 = Indicates setting of any Error Bit in the Error and Status Register 0 = No such occurrence</p>

**Table 301. Fault confinement state**

Value	Meaning
00	Error Active
01	Error Passive
1X	Bus Off

#### 25.4.4.9 Interrupt Masks 2 Register (IMASK2)

This register allows any number of a range of 32 Message Buffer Interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding IFLAG2 bit is set).

Offset: 0x0024

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	63M	62M	61M	60M	59M	58M	57M	56M	55M	54M	53M	52M	51M	50M	49M	48M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	47M	46M	45M	44M	43M	42M	41M	40M	39M	38M	37M	36M	35M	34M	33M	32M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 318. Interrupt Masks 2 Register (IMASK2)

Table 302. MASK2 field descriptions

Field	Description
BUF <sub>n</sub> M	<p>Buffer MB<sub>n</sub> Mask</p> <p>Each bit enables or disables the respective FlexCAN Message Buffer (MB32 to MB63) Interrupt.</p> <p>1 = The corresponding buffer Interrupt is enabled                      0 = The corresponding buffer Interrupt is disabled</p> <p><b>Note:</b> Setting or clearing a bit in the IMASK2 Register can assert or negate an interrupt request, if the corresponding IFLAG2 bit is set.</p>

25.4.4.10 Interrupt Masks 1 Register (IMASK1)

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding IFLAG1 bit is set).

Offset: 0x0028

Access: Read/write

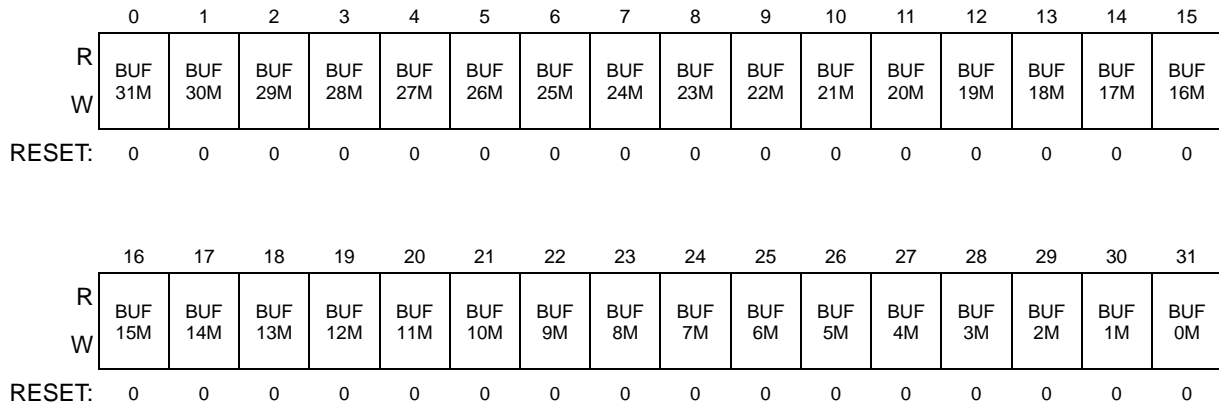


Figure 319. Interrupt Masks 1 Register (IMASK1)

Table 303. IMASK1 field descriptions

Field	Description
BUF <sub>n</sub> M	<p>Buffer MB<sub>n</sub> Mask</p> <p>Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt.</p> <p>1 = The corresponding buffer Interrupt is enabled 0 = The corresponding buffer Interrupt is disabled</p> <p><b>Note:</b> Setting or clearing a bit in the IMASK1 Register can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set.</p>

25.4.4.11 Interrupt Flags 2 Register (IFLAG2)

This register defines the flags for 32 Message Buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG2 bit. If the corresponding IMASK2 bit is set, an interrupt will be generated. The interrupt flag must be cleared by writing it to '1'. Writing '0' has no effect.

When the AEN bit in the MCR is set (Abort enabled), while the IFLAG2 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

Offset: 0x002C

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	63I	62I	61I	60I	59I	58I	57I	56I	55I	54I	53I	52I	51I	50I	49I	48I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	47I	46I	45I	44I	43I	42I	41I	40I	39I	38I	37I	36I	35I	34I	33I	32I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 320. Interrupt Flags 2 Register (IFLAG2)

Table 304. IFLAG2 field descriptions

Field	Description
BUF <sub>nI</sub>	Buffer MB <sub>n</sub> Interrupt Each bit flags the respective FlexCAN Message Buffer (MB32 to MB63) interrupt. 1 = The corresponding buffer has successfully completed transmission or reception 0 = No such occurrence

25.4.4.12 Interrupt Flags 1 Register (IFLAG1)

This register defines the flags for 32 Message Buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to '1'. Writing '0' has no effect.

When the MCR[AEN] bit is set (Abort enabled), while the IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the MCR[FEN] bit is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I - BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Offset: 0x002C

Access: Read/write

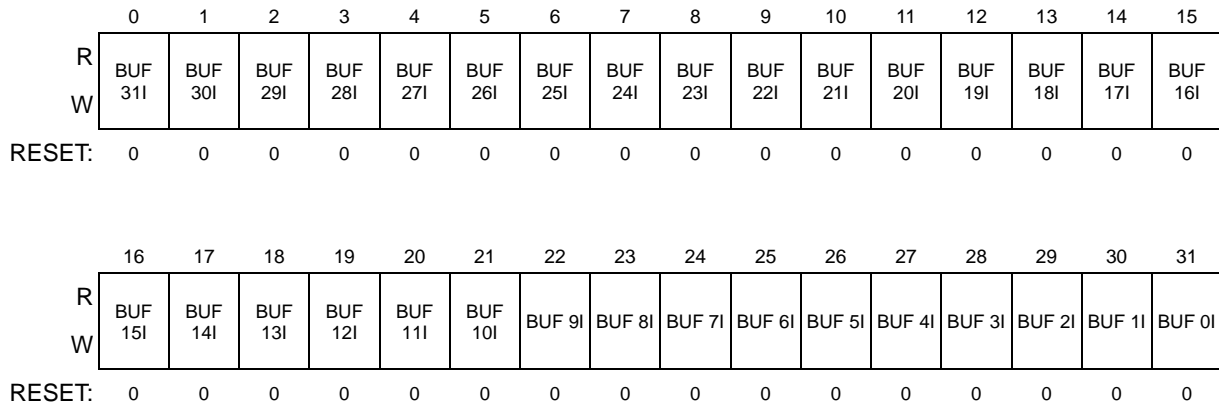


Figure 321. Interrupt Flags 1 Register (IFLAG1)

Table 305. IFLAG1 field descriptions

Field	Description
BUF31I–BUF8I	Buffer MB <sub>n</sub> Interrupt Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt. 1 = The corresponding MB has successfully completed transmission or reception 0 = No such occurrence
BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 1 = MB7 completed transmission/reception or FIFO overflow 0 = No such occurrence
BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 1 = MB6 completed transmission/reception or FIFO almost full 0 = No such occurrence
BUF5I	Buffer MB5 Interrupt or “Frames available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 1 = MB5 completed transmission/reception or frames available in the FIFO 0 = No such occurrence
BUF4I–BUF0I	Buffer MB <sub>i</sub> Interrupt or “reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 1 = Corresponding MB completed transmission/reception 0 = No such occurrence

25.4.4.13 Rx Individual Mask Registers (RXIMR0–RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available Message Buffer, providing ID masking capability on a per Message Buffer basis. When the FIFO is enabled



(FEN bit in MCR is set), the first 8 Mask Registers apply to the 8 elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the BCC bit in the MCR Register is negated, any read or write operation to these registers results in access error.

Offsets: 0x0880–0x097F (64 registers0

Access: Read/write

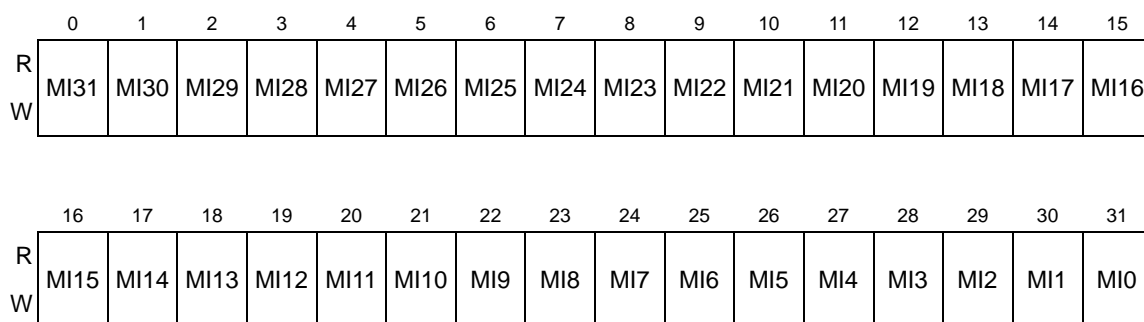


Figure 322. Rx Individual Mask Registers (RXIMR0–RXIMR63)

Table 306. RXIMR0–RXIMR63 field descriptions

Field	Description
MI $n$	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 1 = The corresponding bit in the filter is checked against the one received 0 = The corresponding bit in the filter is “don’t care”

## 25.5 Functional description

### 25.5.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of up to 64 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see [Section 25.4.2: Message Buffer Structure](#)). The memory corresponding to the first 8 MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to 8 extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a ‘0000’ code is inactive (refer to [Table 291](#)). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to [Table 292](#)).

An MB programmed with ‘0000’, ‘1000’ (inactive), or ‘1001’ (abort) will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see [Section 25.5.7.2: Message Buffer Deactivation](#)).

### 25.5.2 Local Priority Transmission

The term local priority refers to the priority of transmit messages of the host node. This allows increased control over the priority mechanism for transmitting messages. [Figure 309](#) shows the placement of PRIO in the ID part of the message buffer.

An additional 3-bit field (PRIO) in the long-word ID part of the message buffer structure has been added for local priority determination. They are prefixed to the regular ID to define the transmission priority. These bits are not transmitted and are intended only for Tx buffers.

Perform the following to use the local priority feature:

1. Set the LPRIO\_EN bit in the CANx\_MCR.
2. Write the additional PRIO bits in the ID long-word of Tx message buffers when configuring the Tx buffers.

With this extended ID concept, the arbitration process is based on the full 32-bit word. However, the actual transmitted ID continues to have 11 bits for standard frames and 29 bits for extended frames.

### 25.5.3 Transmit process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

1. If the MB is active (transmission pending), write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see [Section 25.5.7.1: Transmission Abort Mechanism](#)). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see [Section 25.5.7.2: Message Buffer Deactivation](#)).
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step four (see [Table 291](#) and [Table 292](#) in [Section 25.4.2: Message Buffer Structure](#)). When the Abort feature is enabled (AEN in MCR is asserted), after the Interrupt Flag is asserted for a



MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

### 25.5.4 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID<sup>(s)</sup> or the lowest MB number or the highest priority, depending on the LBUF and LPRIO\_EN bits on the Control Register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any MB
- Upon leaving Freeze Mode

When LBUF is asserted, the LPRIO\_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO\_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO\_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to eight data bytes, even if the DLC (Data Length Code) value is bigger.

### 25.5.5 Receive process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

---

s. Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

1. If the MB has a pending transmission, write an ABORT code ('1001') to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see [Section 25.5.7.1: Transmission Abort Mechanism](#)). If backwards compatibility is desired (AEN in MCR negated), just write '1000' to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see [Section 25.5.7.2: Message Buffer Deactivation](#)). If the MB already programmed as a receiver, just write '0000' to the Code field of the Control and Status word to keep the MB inactive.
2. Write the ID word
3. Write '0100' to the Code field of the Control and Status word to activate the MB

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see [Table 291](#) and [Table 292](#) in [Section 25.4.2: Message Buffer Structure](#))
- A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see [Section 25.5.7: Data coherence](#)).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in [Table 291](#). If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: *never do polling by reading directly the C/S word of the MBs. Instead, read the IFLAG registers.*

Note that the received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX\_DIS bit in the MCR is not asserted. If SRX\_DIS is asserted, FlexCAN will not store

frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze Mode (see [Section 25.5.8: Rx FIFO](#)). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

### 25.5.6 Matching process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see [Section 25.5.7.3: Message Buffer Lock Mechanism](#))
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to [Table 291](#) and [Table 292](#)). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see [Section 25.5.7.3: Message Buffer Lock Mechanism](#)).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in MCR is negated, the matching algorithm stops at the first MB with a matching ID that it finds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to [Section 25.4.4.13: Rx Individual Mask Registers \(RXIMR0–RXIMR63\)](#). During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the MCR Register is negated.

## 25.5.7 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in [Section 25.5.3: Transmit process](#) and [Section 25.5.5: Receive process](#). Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

### 25.5.7.1 Transmission Abort Mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written

into the Code field, the interrupt flag is set in the IFLAG and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE=1001) or it was transmitted (CODE=1000).

### 25.5.7.2 Message Buffer Deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued

and the Code field is not updated. In order to avoid this situation, the abort procedures described in [Section 25.5.7.1: Transmission Abort Mechanism](#) should be used.

### 25.5.7.3 Message Buffer Lock Mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

*Note:* *The locking mechanism only applies to Rx MBs which have a code different than INACTIVE ('0000') or EMPTY<sup>t</sup> ('0100'). Also, Tx MBs can not be locked.*

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

*Note:* *If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.*

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

### 25.5.8 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see [Section 25.4.3: Rx FIFO structure](#)). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

---

t. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior will be honored when the BCC bit is negated.

The FIFO can store up to 6 frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when five frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of eight 32-bit registers that can be configured to one of the following formats (see also [Section 25.4.3: Rx FIFO structure](#)):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

*Note:* A chosen format is applied to all eight registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (RXIMR0 - RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 is affected by RX15MASK and the other elements (0 to 5) are affected by RXGMASK.

## 25.5.9 CAN Protocol Related Features

### 25.5.9.1 Remote Frames

Remote frame is a special kind of frame. The user can program a MB to be a Request Remote Frame by writing the MB as Transmit with the RTR bit set to '1'. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the Code field '1010'. If there is a matching ID, then this MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit a Remote Frame as a response.

A received Remote Request Frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request Frame was received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in MCR), FlexCAN will not generate an automatic response for Remote Request Frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU.

Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

### 25.5.9.2 Overload Frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

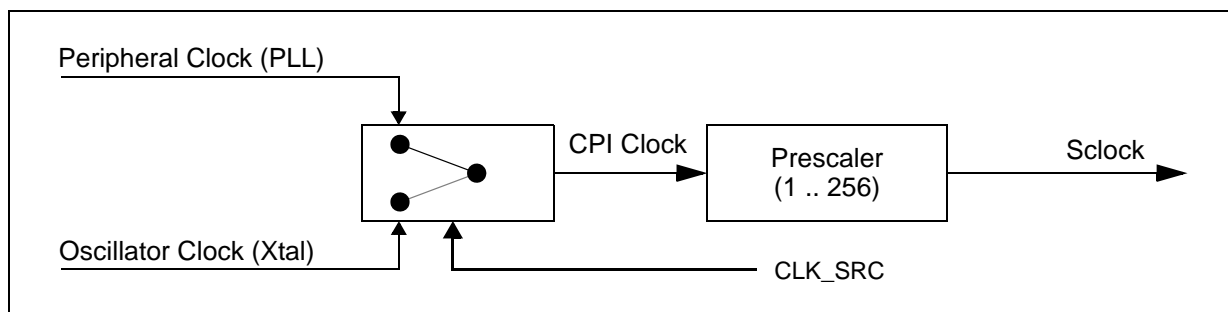
### 25.5.9.3 Time Stamp

The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of “move-in” in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in [Section 25.4.4.2: Control Register \(CTRL\)](#).

### 25.5.9.4 Protocol Timing

[Figure 323](#) shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) sub-module. The clock source bit (CLK\_SRC) in the CTRL Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).



**Figure 323. CAN Engine Clocking Scheme**

The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

*Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, the CLK\_SRC bit in the CTRL Register has no effect on the module operation.*

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit



timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See [Section 25.4.4.2: Control Register \(CTRL\)](#).

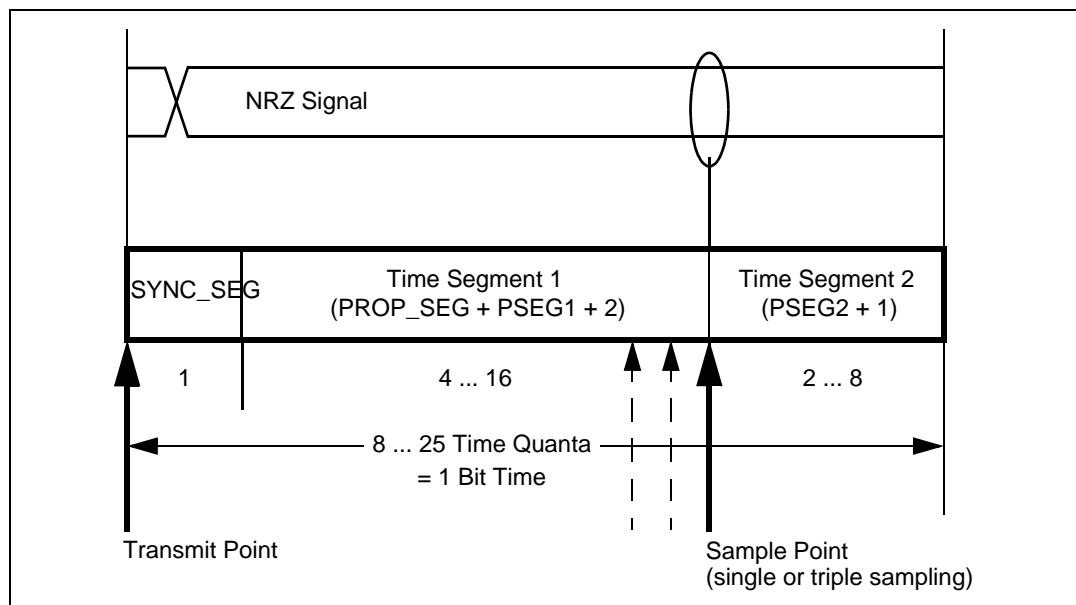
The PRESDIV field controls a prescaler that generates the Serial Clock (Sclock), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

$$f_{Tq} = \frac{f_{CANCLK}}{\text{Prescaler value}}$$

A bit time is subdivided into three segments<sup>(u)</sup> (reference [Figure 324](#) and [Table 307](#)):

- SYNC\_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL Register so that their sum (plus 2) is in the range of 4 to 16 time quanta
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL Register (plus 1) to be 2 to 8 time quanta long

$$\text{Bit Rate} = \frac{f_{Tq}}{\text{number of Time Quanta}}$$



**Figure 324. Segments within the Bit Time**

u. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

**Table 307. Time Segment Syntax**

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 308 gives an overview of the CAN compliant segment settings and the related parameter values.

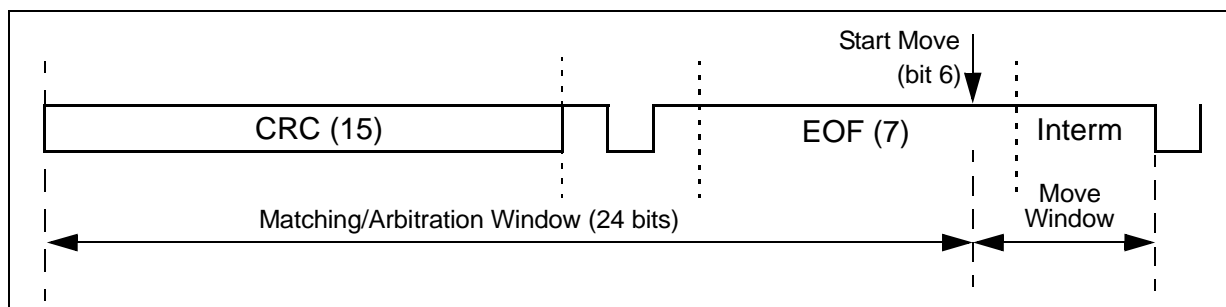
**Table 308. Bosch CAN 2.0B standard compliant bit time segment settings**

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5–10	2	1–2
4–11	3	1–3
5–12	4	1–4
6–13	5	1–4
7–14	6	1–4
8–15	7	1–4
9–16	8	1–4

*Note:* Other combinations of Time Segment 1 and Time Segment 2 can be valid. It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

**25.5.9.5 Arbitration and Matching Timing**

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in Figure 325.



**Figure 325. Arbitration, Match and Move Time Windows**

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in [Table 308](#)
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, i.e., the PLL can not be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in [Table 309](#)

**Table 309. Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate**

Number of Message Buffers	Minimum Ratio
16	8
32	8
64	16

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in [Table 309](#) can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRES DIV, PROPSEG, PSEG1, PSEG2). As an example, taking the case of 64 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRES DIV + 1) should be at least 2. For prescaler factor equal to one and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

## 25.5.10 Modes of operation details

### 25.5.10.1 Freeze mode

This mode is entered by asserting the HALT bit in the MCR Register or when the MCU is put into Debug Mode. In both cases it is also necessary that the FRZ bit is asserted in the MCR Register and the module is not in a low-power mode (Disable mode). When Freeze Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off or Idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOT\_RDY and FRZ\_ACK bits in MCR

After requesting Freeze Mode, the user must wait for the FRZ\_ACK bit to be asserted in MCR before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible.

Exiting Freeze Mode is done in one of the following ways:

- CPU negates the FRZ bit in the MCR Register
- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

### 25.5.10.2 Module Disable mode

This low power mode is entered when the MCR[MDIS] bit is asserted. If the module is disabled during Freeze Mode, it requests to disable the clocks to the CAN Protocol Interface (CPI) and Message Buffer Management (MBM) sub-modules, sets the LPM\_ACK bit and negates the FRZ\_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT\_RDY and LPM\_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM\_ACK bit.

## 25.5.11 Interrupts

The module can generate up to 70 interrupt sources (64 interrupts due to message buffers and 6 interrupts due to Ored interrupts from MBs, Bus Off, Error, Tx Warning, Rx Warning and Wake Up). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFLAG Registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to '1' (unless another interrupt is generated at the same time).

*Note: It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.*

If the Rx FIFO is enabled (bit FEN on MCR set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the IFLAG1 becomes the "FIFO Overflow" flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the "Frames Available in FIFO flag" and bits 4-0 are unused. See [Section 25.4.4.12: Interrupt Flags 1 Register \(IFLAG1\)](#) for more information.

A combined interrupt for all MBs is also generated by an Or of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the IFLAG Registers to determine which MB caused the interrupt.

The other 5 interrupt sources (Bus Off, Error, Tx Warning, Rx Warning and Wake Up) generate interrupts like the MB ones, and can be read from the Error and Status Register.

The Bus Off, Error, Tx Warning and Rx Warning interrupt mask bits are located in the Control Register, and the Wake-Up interrupt mask bit is located in the MCR.

## 25.5.12 Bus interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when the BCC bit in MCR is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the Mask Registers space would be from 0x0884 to 0x097F.

*Note:* Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

## 25.6 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

### 25.6.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to [Table 288](#) to see what registers are affected by soft reset)
- SOFT\_RST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT\_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLK\_SRC bit) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is un-synchronized to the CAN bus, the HALT and FRZ bits in MCR Register are set, the internal state machines are disabled and the FRZ\_ACK and NOT\_RDY bits in the MCR Register are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN

frames. Note that the Message Buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see [Section 25.5.10.1: Freeze mode](#)). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the Module Configuration Register
  - Enable the individual filtering per MB and reception queue features by setting the BCC bit
  - Enable the warning interrupts by setting the WRN\_EN bit
  - If required, disable frame self reception by setting the SRX\_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO\_EN bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRES DIV field
  - Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts), in CTRL Register (for Bus Off and Error interrupts) and in MCR Register for Wake-Up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

## 25.6.2 FlexCAN Addressing and RAM size configurations

There are three RAM configurations that can be implemented within the FlexCAN module. The possible configurations are:

- For 16 MBs: 288 bytes for MB memory and 64 bytes for Individual Mask Registers
- For 32 MBs: 544 bytes for MB memory and 128 bytes for Individual Mask Registers
- For 64 MBs: 1056 bytes for MB memory and 256 bytes for Individual Mask Registers

In each configuration the user can program the maximum number of MBs that will take part in the matching and arbitration processes using the MAXMB field in the MCR Register. For 16 MB configuration, MAXMB can be any number between 0–15. For 32 MB configuration, MAXMB can be any number between 0–31. For 64 MB configuration, MAXMB can be any number between 0–63.

## 26 Deserial Serial Peripheral Interface (DSPI)

### 26.1 Introduction

This chapter describes the Deserial Serial Peripheral Interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

The SPC560B54/6x has six identical DSPI modules (DSPI\_0 – DSPI\_5). The “x” appended to signal names signifies the module to which the signal applies. Thus CS0\_x specifies that the CS0 signal applies to DSPI module 0, 1, etc.

A block diagram of the DSPI is shown in [Figure 326](#).

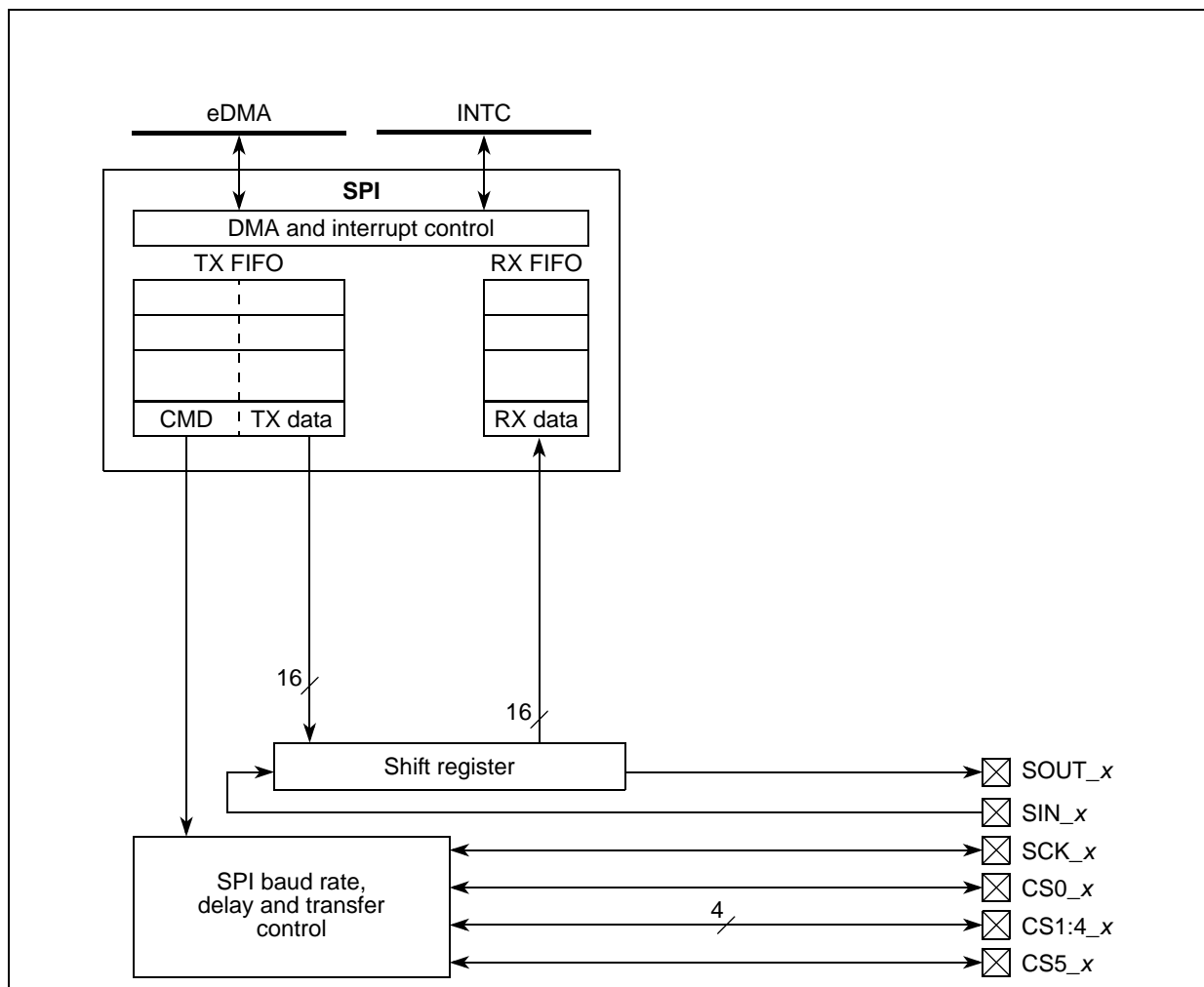


Figure 326. DSPI block diagram

The register content is transmitted using an SPI protocol.

For queued operations the SPI queues reside in internal SRAM which is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

Figure 327 shows a DSPI with external queues in internal SRAM.

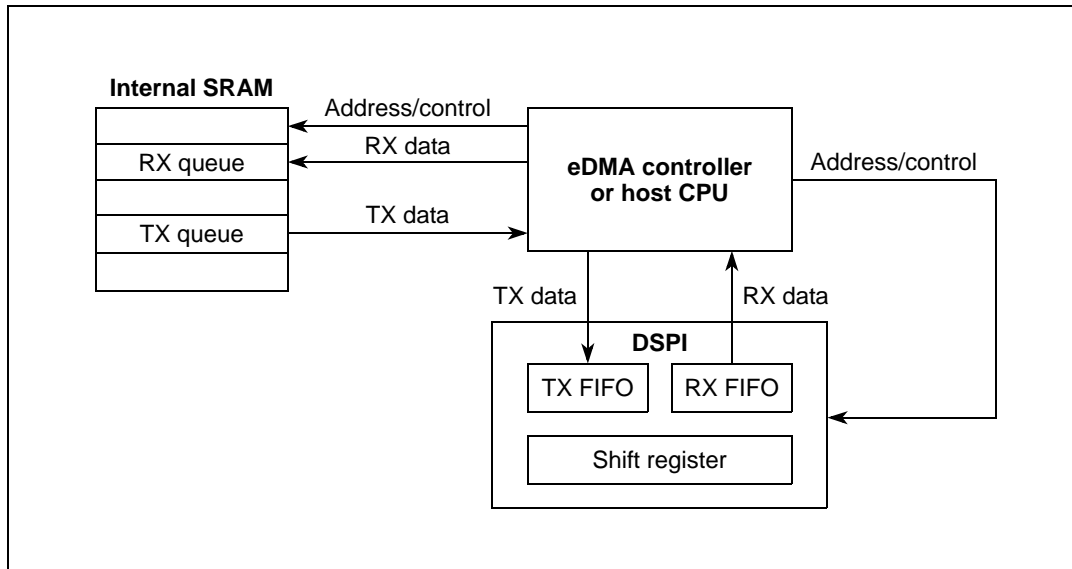


Figure 327. DSPI with queues and eDMA

## 26.2 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of four entries
- Visibility into TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - 6 clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays
    - CS to SCK delay
    - SCK to CS delay
    - Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- Up to 6 peripheral chip selects, expandable to 64 with external demultiplexer



- Deglitching support for up to 32 peripheral chip selects with external demultiplexer
- Two DMA conditions for SPI queues residing in RAM or flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 6 interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - RX FIFO is not empty (RFDF)
  - FIFO overrun (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF) or (TFUF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (SCK)

## 26.3 Modes of operation

The DSPI has five modes of operation. These modes can be divided into two categories:

- Module-specific: Master, Slave, and Module Disable modes
- MCU-specific: External Stop and Debug modes

The module-specific modes are entered by host software writing to a register. The MCU-specific modes are controlled by signals external to the DSPI. An MCU-specific mode is a mode that the entire device may enter, in parallel to the DSPI being in one of its module-specific modes.

### 26.3.1 Master mode

Master mode allows the DSPI to initiate and control serial communication. In this mode the SCK, CS<sub>n</sub> and SOUT signals are controlled by the DSPI and configured as outputs.

For more information, see [Section 26.6.1.1: Master mode](#).

### 26.3.2 Slave mode

Slave mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in slave mode. In slave mode, the SCK signal and the CS<sub>0\_x</sub> signal are configured as inputs and provided by a bus master. CS<sub>0\_x</sub> must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU\_PCR must be set (SIU\_PCR [WPE = 1], [WPS = 1]).

For more information, see [Section 26.6.1.2: Slave mode](#).

### 26.3.3 Module Disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI<sub>x</sub>\_MCR is set.

For more information, see [Section 26.6.1.3: Module Disable mode](#).

### 26.3.4 Debug mode

Debug mode is used for system development and debugging. If the device enters debug mode while the FRZ bit in the DSPI<sub>x</sub>\_MCR is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI.

For more information, see [Section 26.6.1.4: Debug mode](#).

## 26.4 External signal description

### 26.4.1 Signal overview

[Table 310](#) lists off-chip DSPI signals.

**Table 310. Signal properties**

Name	I/O type	Function	
		Master mode	Slave mode
CS0_x	Output / input	Peripheral chip select 0	Slave select
CS1:3_x	Output	Peripheral chip select 1–3	Unused <sup>(1)</sup>
CS4_x	Output	Peripheral chip select 4	Master trigger
CS5_x	Output	Peripheral chip select 5 / Peripheral chip select strobe	Unused <sup>(1)</sup>
SIN_x	Input	Serial data in	Serial data in
SOUT_x	Output	Serial data out	Serial data out
SCK_x	Output / input	Serial clock (output)	Serial clock (input)

1. The SIUL allows you to select alternate pin functions for the device.

### 26.4.2 Signal names and descriptions

#### 26.4.2.1 Peripheral Chip Select / Slave Select (CS0\_x)

In master mode, the CS0\_x signal is a peripheral chip select output that selects the slave device to which the current transmission is intended.

In slave mode, the CS0\_x signal is a slave select input signal that allows an SPI master to select the DSPI as the target for transmission. CS0\_x must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU\_PCR must be set (SIU\_PCR [WPE = 1], [WPS = 1]).

Set the IBE and OBE bits in the SIU\_PCR for all CS0\_x pins when the DSPI chip select or slave select primary function is selected for that pin. When the pin is used for DSPI master mode as a chip select output, set the OBE bit. When the pin is used in DSPI slave mode as a slave select input, set the IBE bit.

### 26.4.2.2 Peripheral Chip Selects 1–3 (CS1:3\_x)

CS1:3\_x are peripheral chip select output signals in master mode. In slave mode these signals are not used.

### 26.4.2.3 Peripheral Chip Select 4 (CS4\_x)

CS4\_x is a peripheral chip select output signal in master mode.

### 26.4.2.4 Peripheral Chip Select 5 / Peripheral Chip Select Strobe (CS5\_x)

CS5\_x is a peripheral chip select output signal. When the DSPI is in master mode and PCSSE bit in the DSPIx\_MCR is cleared, the CS5\_x signal is used to select the slave device that receives the current transfer.

CS5\_x is a strobe signal used by external logic for deglitching of the CS signals. When the DSPI is in master mode and the PCSSE bit in the DSPIx\_MCR is set, the CS5\_x signal indicates the timing to decode CS0:4\_x signals, which prevents glitches from occurring.

CS5\_x is not used in slave mode.

### 26.4.2.5 Serial Input (SIN\_x)

SIN\_x is a serial data input signal.

### 26.4.2.6 Serial Output (SOUT\_x)

SOUT\_x is a serial data output signal.

### 26.4.2.7 Serial Clock (SCK\_x)

SCK\_x is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK\_x is an input from an external bus master.

## 26.5 Memory map and register description

### 26.5.1 Memory map

[Table 311](#) shows the DSPI memory map.

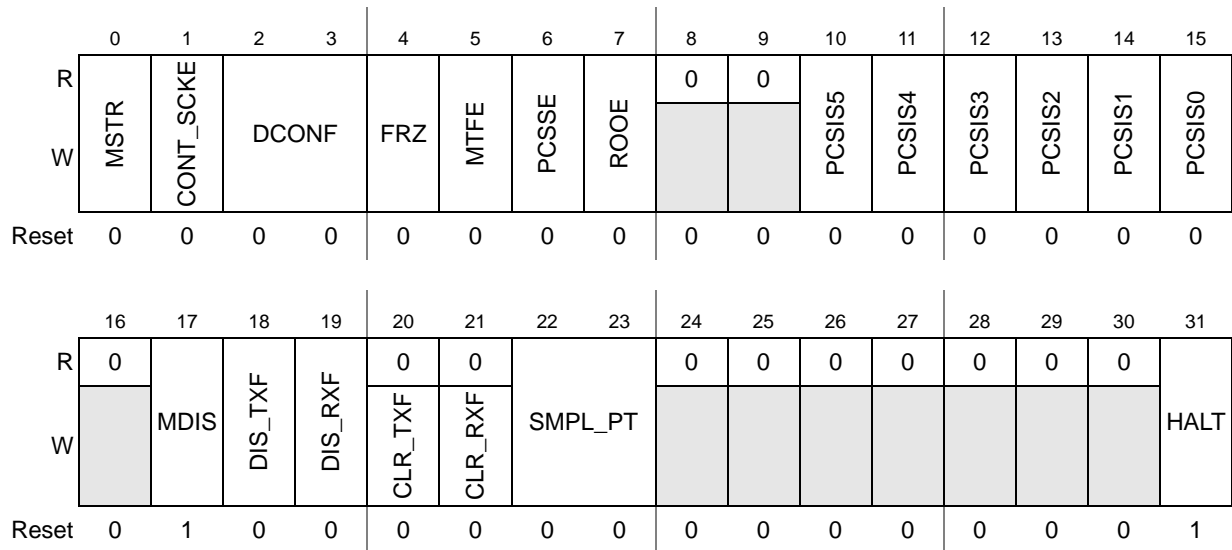
Table 311. DSPI memory map

Base addresses:		
0xFFFF9_0000 (DSPI_0)		
0xFFFF9_4000 (DSPI_1)		
0xFFFF9_8000 (DSPI_2)		
0xFFFF9_C000 (DSPI_3)		
0xFFFFA_0000 (DSPI_4)		
0xFFFFA_4000 (DSPI_5)		
Address offset	Register	Location
0x00	DSPI Module Configuration Register (DSPIx_MCR)	<a href="#">on page 644</a>
0x04	Reserved	
0x08	DSPI Transfer Count Register (DSPIx_TCR)	<a href="#">on page 647</a>
0x0C	DSPI Clock and Transfer Attributes Register 0 (DSPIx_CTAR0)	<a href="#">on page 648</a>
0x10	DSPI Clock and Transfer Attributes Register 1 (DSPIx_CTAR1)	<a href="#">on page 648</a>
0x14	DSPI Clock and Transfer Attributes Register 2 (DSPIx_CTAR2)	<a href="#">on page 648</a>
0x18	DSPI Clock and Transfer Attributes Register 3 (DSPIx_CTAR3)	<a href="#">on page 648</a>
0x1C	DSPI Clock and Transfer Attributes Register 4 (DSPIx_CTAR4)	<a href="#">on page 648</a>
0x20	DSPI Clock and Transfer Attributes Register 5 (DSPIx_CTAR5)	<a href="#">on page 648</a>
0x24–0x28	Reserved	
0x2C	DSPI Status Register (DSPIx_SR)	<a href="#">on page 656</a>
0x30	DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)	<a href="#">on page 658</a>
0x34	DSPI Push TX FIFO Register (DSPIx_PUSHR)	<a href="#">on page 660</a>
0x38	DSPI Pop RX FIFO Register (DSPIx_POPR)	<a href="#">on page 662</a>
0x3C	DSPI Transmit FIFO Register 0 (DSPIx_TXFR0)	<a href="#">on page 662</a>
0x40	DSPI Transmit FIFO Register 1 (DSPIx_TXFR1)	<a href="#">on page 662</a>
0x44	DSPI Transmit FIFO Register 2 (DSPIx_TXFR2)	<a href="#">on page 662</a>
0x48	DSPI Transmit FIFO Register 3 (DSPIx_TXFR3)	<a href="#">on page 662</a>
0x4C–0x78	Reserved	
0x7C	DSPI Receive FIFO Register 0 (DSPIx_RXFR0)	<a href="#">on page 663</a>
0x80	DSPI Receive FIFO Register 1 (DSPIx_RXFR1)	<a href="#">on page 663</a>
0x84	DSPI Receive FIFO Register 2 (DSPIx_RXFR2)	<a href="#">on page 663</a>
0x88	DSPI Receive FIFO Register 3 (DSPIx_RXFR3)	<a href="#">on page 663</a>

### 26.5.2 DSPI Module Configuration Register (DSPIx\_MCR)

The DSPIx\_MCR contains bits which configure attributes of the DSPI operation. The values of the HALT and MDIS bits can be changed at any time, but their effect begins on the next frame boundary. The HALT and MDIS bits in the DSPIx\_MCR are the only bit values software can change while the DSPI is running.

Offset: 0x00 Access: Read/write



**Figure 328. DSPI Module Configuration Register (DSPIx\_MCR)**

**Table 312. DSPIx\_MCR field descriptions**

Field	Description										
MSTR	Master/slave mode select Configures the DSPI for master mode or slave mode.  0 DSPI is in slave mode 1 DSPI is in master mode										
CONT_SCKE	Continuous SCK enable Enables the serial communication clock (SCK) to run continuously. See <a href="#">Section 26.6.6: Continuous serial communications clock</a> , for details.  0 Continuous SCK disabled 1 Continuous SCK enabled										
DCONF	DSPI configuration The following table lists the DCONF values for the various configurations. <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>DCONF</th> <th>Configuration</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>SPI</td> </tr> <tr> <td>01</td> <td>Invalid value</td> </tr> <tr> <td>10</td> <td>Invalid value</td> </tr> <tr> <td>11</td> <td>Invalid value</td> </tr> </tbody> </table>	DCONF	Configuration	00	SPI	01	Invalid value	10	Invalid value	11	Invalid value
DCONF	Configuration										
00	SPI										
01	Invalid value										
10	Invalid value										
11	Invalid value										

Table 312. DSPIx\_MCR field descriptions (continued)

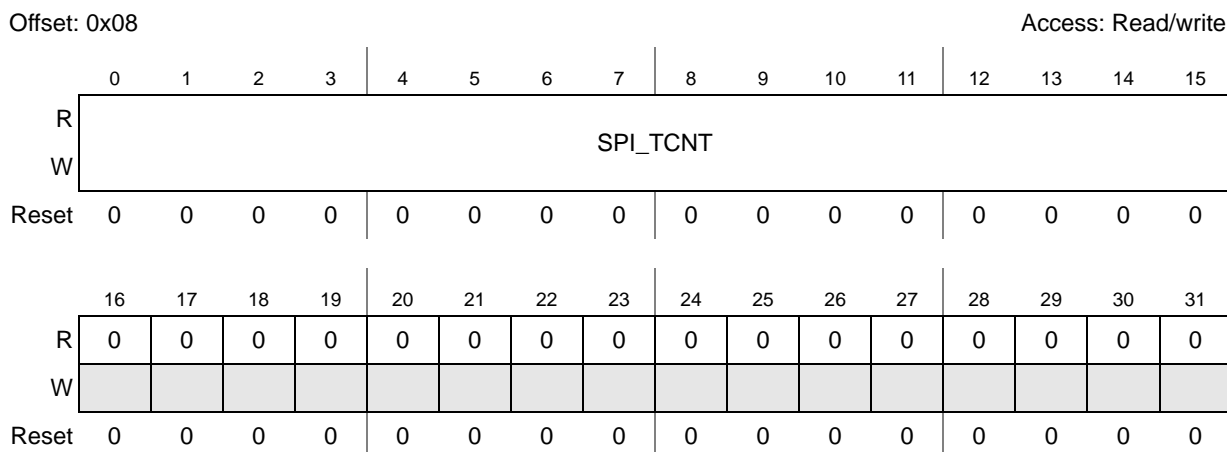
Field	Description
FRZ	<p>Freeze</p> <p>Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode.</p> <p>0 Do not halt serial transfers 1 Halt serial transfers</p>
MTFE	<p>Modified timing format enable</p> <p>Enables a modified transfer format to be used. See <a href="#">Section 26.6.5.4: Modified SPI transfer format (MTFE = 1, CPHA = 1)</a>, for more information.</p> <p>0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled</p>
PCSSE	<p>Peripheral chip select strobe enable</p> <p>Enables the CS5_x to operate as a CS strobe output signal. See <a href="#">Section 26.6.4.5: Peripheral chip select strobe enable (CS5_x)</a>, for more information.</p> <p>0 CS5_x is used as the Peripheral chip select 5 signal 1 CS5_x as an active-low CS strobe signal</p>
ROOE	<p>Receive FIFO overflow overwrite enable</p> <p>Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is ignored or put in the shift register.</p> <p>If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored. See <a href="#">Section 26.6.7.6: Receive FIFO Overflow Interrupt Request (RFOF)</a>, for more information.</p> <p>0 Incoming data is ignored 1 Incoming data is put in the shift register</p>
PCSI <sub>n</sub>	<p>Peripheral chip select inactive state</p> <p>Determines the inactive state of the CS0_x signal. CS0_x must be configured as inactive high for slave mode operation.</p> <p>0 The inactive state of CS0_x is low 1 The inactive state of CS0_x is high</p>
MDIS	<p>Module disable</p> <p>Allows the clock to stop to the non-memory mapped logic in the DSPI, effectively putting the DSPI in a software controlled power-saving state. See <a href="#">Section 26.6.8: Power saving features</a> for more information.</p> <p>0 Enable DSPI clocks 1 Allow external logic to disable DSPI clocks</p>

**Table 312. DSPIx\_MCR field descriptions (continued)**

Field	Description										
DIS_TXF	<p>Disable transmit FIFO</p> <p>Enables and disables the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See <a href="#">Section 26.6.3.3: FIFO disable operation</a> for details.</p> <p>0 TX FIFO is enabled 1 TX FIFO is disabled</p>										
DIS_RXF	<p>Disable receive FIFO</p> <p>Enables and disables the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See <a href="#">Section 26.6.3.3: FIFO disable operation</a> for details.</p> <p>0 RX FIFO is enabled 1 RX FIFO is disabled</p>										
CLR_TXF	<p>Clear TX FIFO. CLR_TXF is used to flush the TX FIFO. Writing a '1' to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero.</p> <p>0 Do not clear the TX FIFO Counter 1 Clear the TX FIFO Counter</p>										
CLR_RXF	<p>Clear RX FIFO. CLR_RXF is used to flush the RX FIFO. Writing a '1' to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero.</p> <p>0 Do not clear the RX FIFO Counter 1 Clear the RX FIFO Counter</p>										
SMPL_PT	<p>Sample point</p> <p>Allows the host software to select when the DSPI master samples SIN in modified transfer format. <a href="#">Figure 343</a> shows where the master can sample the SIN pin. The following table lists the delayed sample points.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>SMPL_PT</th> <th>Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>0</td> </tr> <tr> <td>01</td> <td>1</td> </tr> <tr> <td>10</td> <td>2</td> </tr> <tr> <td>11</td> <td>Reserved</td> </tr> </tbody> </table>	SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x	00	0	01	1	10	2	11	Reserved
SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x										
00	0										
01	1										
10	2										
11	Reserved										
HALT	<p>Halt</p> <p>Provides a mechanism for software to start and stop DSPI transfers. See <a href="#">Section 26.6.2: Start and stop of DSPI transfers</a>, for details on the operation of this bit.</p> <p>0 Start transfers 1 Stop transfers</p>										

### 26.5.3 DSPI Transfer Count Register (DSPIx\_TCR)

The DSPIx\_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. The user must not write to the DSPIx\_TCR while the DSPI is running.



**Figure 329. DSPI Transfer Count Register (DSPIx\_TCR)**

**Table 313. DSPIx\_TCR field descriptions**

Field	Description
SPI_TCNT	SPI transfer counter Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented every time the last bit of an SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter 'wraps around,' incrementing the counter past 65535 resets the counter to zero.

### 26.5.4 DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx\_CTARn)

The DSPI modules each contain six clock and transfer attribute registers (DSPIx\_CTARn) which are used to define different transfer attribute configurations. Each DSPIx\_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB or LSB first

DSPIx\_CTARs support compatibility with the QSPI module in the SPC560B54/6x family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPIx\_CTAR that contains the transfer's attributes. Do not write to the DSPIx\_CTARs while the DSPI is running.

In master mode, the DSPIx\_CTARn registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bit fields in the DSPIx\_CTAR0 and DSPIx\_CTAR1 registers are used to set the slave transfer attributes. See the individual bit descriptions for details on which bits are used in slave modes.

When the DSPI is configured as an SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPIx\_CTAR registers is used on a per-frame basis. When the DSPI is configured as an SPI bus slave, the DSPIx\_CTAR0 register is used.



Offsets: 0x0C–0x20 (6 registers)

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DBR	FMSZ			CPOL	CPHA	LSBFE	PCSSCK		PASC		PDT		PBR		
W	DBR	FMSZ			CPOL	CPHA	LSBFE	PCSSCK		PASC		PDT		PBR		
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSSCK				ASC				DT				BR			
W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 330. DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx\_CTARn)

Table 314. DSPIx\_CTARn field descriptions

Field	Descriptions
DBR	<p>Double Baud Rate</p> <p>The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in Master Mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in <a href="#">Table 321</a>. See the BR[0:3] field description for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the Continuous SCK Enable or the Modified Timing Format Enable bits should be set.</p> <p>0 The baud rate is computed normally with a 50/50 duty cycle                      1 The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler</p>
FMSZ	<p>Frame Size</p> <p>The FMSZ field selects the number of bits transferred per frame. The FMSZ field is used in Master Mode and Slave Mode. <a href="#">Table 322</a> lists the frame size encodings.</p>
CPOL	<p>Clock Polarity</p> <p>The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge.</p> <p>0 The inactive state value of SCK is low                      1 The inactive state value of SCK is high</p>
CPHA	<p>Clock Phase</p> <p>The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock phase settings. Continuous SCK is only supported for CPHA = 1.</p> <p>0 Data is captured on the leading edge of SCK and changed on the following edge                      1 Data is changed on the leading edge of SCK and captured on the following edge</p>

**Table 314. DSPIx\_CTARn field descriptions (continued)**

Field	Descriptions										
LSBFE	<p>LSB First</p> <p>The LSBFE bit selects if the LSB or MSB of the frame is transferred first. This bit is only used in Master Mode.</p> <p>0 Data is transferred MSB first                      1 Data is transferred LSB first</p>										
PCSSCK	<p>PCS to SCK Delay Prescaler</p> <p>The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. This field is only used in Master Mode. The table below lists the prescaler values. See the CSSCK field description for details on how to compute the PCS to SCK delay.</p> <table border="1" data-bbox="512 683 1157 920"> <thead> <tr> <th>PCSSCK</th> <th>PCS to SCK delay prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PCSSCK	PCS to SCK delay prescaler value	00	1	01	3	10	5	11	7
PCSSCK	PCS to SCK delay prescaler value										
00	1										
01	3										
10	5										
11	7										
PASC	<p>After SCK Delay Prescaler</p> <p>The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. This field is only used in Master Mode. The table below lists the prescaler values. See the ASC[0:3] field description for details on how to compute the After SCK delay.</p> <table border="1" data-bbox="512 1144 1157 1382"> <thead> <tr> <th>PASC</th> <th>After SCK delay prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PASC	After SCK delay prescaler value	00	1	01	3	10	5	11	7
PASC	After SCK delay prescaler value										
00	1										
01	3										
10	5										
11	7										
PDT	<p>Delay after Transfer Prescaler</p> <p>The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in Master Mode. The table below lists the prescaler values. See the DT[0:3] field description for details on how to compute the delay after transfer.</p> <table border="1" data-bbox="537 1628 1147 1865"> <thead> <tr> <th>PDT</th> <th>Delay after transfer prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PDT	Delay after transfer prescaler value	00	1	01	3	10	5	11	7
PDT	Delay after transfer prescaler value										
00	1										
01	3										
10	5										
11	7										

**Table 314. DSPIx\_CTARn field descriptions (continued)**

Field	Descriptions										
PBR	<p>Baud Rate Prescaler</p> <p>The PBR field selects the prescaler value for the baud rate. This field is only used in Master Mode. The Baud Rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The Baud Rate Prescaler values are listed in the table below. See the BR[0:3] field description for details on how to compute the baud rate.</p> <table border="1" data-bbox="541 539 1118 775"> <thead> <tr> <th>PBR</th> <th>Baud rate prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>2</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PBR	Baud rate prescaler value	00	2	01	3	10	5	11	7
PBR	Baud rate prescaler value										
00	2										
01	3										
10	5										
11	7										
CSSCK	<p>PCS to SCK Delay Scaler</p> <p>The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in Master Mode. The PCS to SCK Delay is the delay between the assertion of PCS and the first edge of the SCK. <a href="#">Table 323</a> list the scaler values. The PCS to SCK Delay is a multiple of the system clock period and it is computed according to the following equation:</p> <p><b>Equation 8</b> <math display="block">t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK</math></p> <p>See <a href="#">Section 26.6.4.2: CS to SCK delay (tCSC)</a>, for more details.</p>										
ASC	<p>After SCK Delay Scaler</p> <p>The ASC field selects the scaler value for the After SCK Delay. This field is only used in Master Mode. The After SCK Delay is the delay between the last edge of SCK and the negation of PCS. <a href="#">Table 324</a> lists the scaler values. The After SCK Delay is a multiple of the system clock period, and it is computed according to the following equation:</p> <p><b>Equation 9</b> <math display="block">t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC</math></p> <p>See <a href="#">Section 26.6.4.3: After SCK delay (tASC)</a>, for more details.</p>										
DT	<p>Delay after Transfer Scaler</p> <p>The DT field selects the Delay after Transfer Scaler. This field is only used in Master Mode. The Delay after Transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. <a href="#">Table 325</a> lists the scaler values. In the Continuous Serial Communications Clock operation the DT value is fixed to one TSCK. The Delay after Transfer is a multiple of the system clock period and it is computed according to the following equation:</p> <p><b>Equation 10</b> <math display="block">t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT</math></p> <p>See <a href="#">Section 26.6.4.4: Delay after transfer (tDT)</a>, for more details.</p>										

**Table 314. DSPIx\_CTARn field descriptions (continued)**

Field	Descriptions
BR	<p>Baud Rate Scaler</p> <p>The BR field selects the scaler value for the baud rate. This field is only used in Master Mode. The prescaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. <a href="#">Table 326</a> lists the Baud Rate Scaler values. The baud rate is computed according to the following equation:</p> <p><b>Equation 11</b>      <math display="block">\text{SCK baud rate} = \frac{f_{\text{SYS}}}{\text{PBR}} \times \frac{1 + \text{DBR}}{\text{BR}}</math></p> <p>See <a href="#">Section 26.6.4.2: CS to SCK delay (tCSC)</a>, for more details.</p>

**Table 315. DSPI SCK duty cycle**

DBR	CPHA	PBR	SCK duty cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

**Table 316. DSPI transfer frame size**

FMSZ	Frame size	FMSZ	Frame size
0000	Reserved	1000	9
0001	Reserved	1001	10
0010	Reserved	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

**Table 317. DSPI PCS to SCK delay scaler**

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to SCK delay scaler value
0000	2	1000	512
0001	4	1001	1024

**Table 317. DSPI PCS to SCK delay scaler (continued)**

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to SCK delay scaler value
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

**Table 318. DSPI After SCK delay scaler**

ASC	After SCK delay scaler value	ASC	After SCK delay scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

**Table 319. DSPI delay after transfer scaler**

DT	Delay after transfer scaler value	DT	Delay after transfer scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

**Table 320. DSPI baud rate scaler**

BR	Baud rate scaler value	BR	Baud rate scaler value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024

**Table 320. DSPI baud rate scaler (continued)**

BR	Baud rate scaler value	BR	Baud rate scaler value
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

**Table 321. DSPI SCK duty cycle**

DBR	CPHA	PBR	SCK duty cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

**Table 322. DSPI transfer frame size**

FMSZ	Frame size	FMSZ	Frame size
0000	Reserved	1000	9
0001	Reserved	1001	10
0010	Reserved	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

**Table 323. DSPI PCS to SCK delay scaler**

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to SCK delay scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048

**Table 323. DSPI PCS to SCK delay scaler (continued)**

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to SCK delay scaler value
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

**Table 324. DSPI After SCK delay scaler**

ASC	After SCK delay scaler value	ASC	After SCK delay scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

**Table 325. DSPI delay after transfer scaler**

DT	Delay after transfer scaler value	DT	Delay after transfer scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

**Table 326. DSPI baud rate scaler**

BR	Baud rate scaler value	BR	Baud rate scaler value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048

Table 326. DSPI baud rate scaler (continued)

BR	Baud rate scaler value	BR	Baud rate scaler value
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

### 26.5.5 DSPI Status Register (DSPIx\_SR)

The DSPIx\_SR contains status and flag bits. The bits are set by the hardware and reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPIx\_SR by writing a '1' to clear it (w1c). Writing a '0' to a flag bit has no effect. This register may not be writable in Module Disable mode due to the use of power saving mechanisms.

Offset: 0x2C

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RDFL	0
W	w1c			w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 331. DSPI Status Register (DSPIx\_SR)

Table 327. DSPIx\_SR field descriptions

Field	Description
TCF	Transfer complete flag Indicates that all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the $t_{ASC}$ delay starts. See <a href="#">Section 26.6.5.1: Classic SPI transfer format (CPHA = 0)</a> for details.  0 Transfer not complete 1 Transfer complete
TXRXS	TX and RX status Reflects the status of the DSPI. See <a href="#">Section 26.6.2: Start and stop of DSPI transfers</a> for information on what clears and sets this bit.  0 TX and RX operations are disabled (DSPI is in STOPPED state) 1 TX and RX operations are enabled (DSPI is in RUNNING state)





Table 327. DSPIx\_SR field descriptions (continued)

Field	Description
EOQF	<p>End of queue flag</p> <p>Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. See <a href="#">Section 26.6.5.1: Classic SPI transfer format (CPHA = 0)</a> for details.</p> <p>When the EOQF bit is set, the TXRXS bit is automatically cleared.</p> <p>0 EOQ is not set in the executing command 1 EOQ bit is set in the executing SPI command</p> <p><b>Note:</b> EOQF does not function in slave mode.</p>
TFUF	<p>Transmit FIFO underflow flag</p> <p>Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master.</p> <p>0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred</p>
TFFF	<p>Transmit FIFO fill flag</p> <p>Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing '1' to it, or an by acknowledgement from the Edam controller when the TX FIFO is full.</p> <p>0 TX FIFO is full 1 TX FIFO is not full</p>
RFOF	<p>Receive FIFO overflow flag</p> <p>Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated.</p> <p>0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred</p>
RFDF	<p>Receive FIFO drain flag</p> <p>Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing '1' to it, or by acknowledgement from the Edam controller when the RX FIFO is empty.</p> <p>0 RX FIFO is empty 1 RX FIFO is not empty</p> <p><b>Note:</b> In the interrupt service routine, RFDF must be cleared only after the DSPIx_POPR register is read.</p>
TXCTR	<p>TX FIFO counter</p> <p>Indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI_PUSH is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.</p>

Table 327. DSPIx\_SR field descriptions (continued)

Field	Description
TXNXTPTR	Transmit next pointer Indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See <a href="#">Section 26.6.3.4: Transmit First In First Out (TX FIFO) buffering mechanism</a> for more details.
RXCTR	RX FIFO counter Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPIx_POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the t <sub>ASC</sub> delay starts. See <a href="#">Section 26.6.5.1: Classic SPI transfer format (CPHA = 0)</a> for details.
POPNXTPTR	Pop next pointer Contains a pointer to the RX FIFO entry that is returned when the DSPIx_POPR is read. The POPNXTPTR is updated when the DSPIx_POPR is read. See <a href="#">Section 26.6.3.5: Receive First In First Out (RX FIFO) buffering mechanism</a> for more details.

### 26.5.6 DSPI DMA / Interrupt Request Select and Enable Register (DSPIx\_RSER)

The DSPIx\_RSER serves two purposes:

- It enables flag bits in the DSPIx\_SR to generate DMA requests or interrupt requests.
- It selects the type of request to generate.

See the bit descriptions for the type of requests that are supported.

Do not write to the DSPIx\_RSER while the DSPI is running.

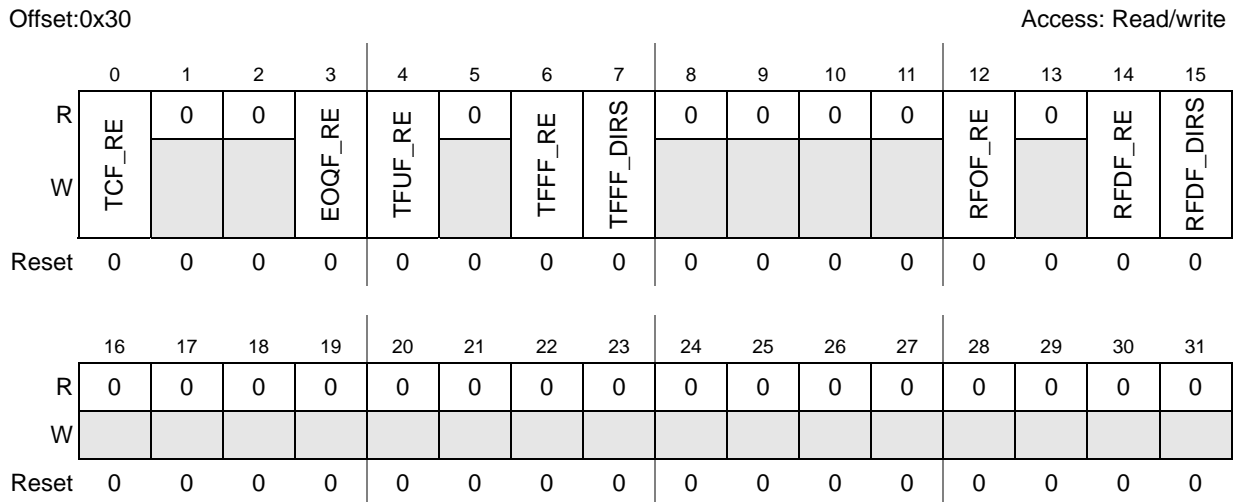


Figure 332. DSPI DMA / Interrupt Request Select and Enable Register (DSPIx\_RSER)

Table 328. DSPIx\_RSER field descriptions

Field	Description
TCF_RE	<p>Transmission complete request enable Enables TCF flag in the DSPIx_SR to generate an interrupt request.</p> <p>0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled</p>
EOQF_RE	<p>DSPI finished request enable Enables the EOQF flag in the DSPIx_SR to generate an interrupt request.</p> <p>0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled</p>
TFUF_RE	<p>Transmit FIFO underflow request enable The TFUF_RE bit enables the TFUF flag in the DSPIx_SR to generate an interrupt request.</p> <p>0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled</p>
TFFF_RE	<p>Transmit FIFO fill request enable Enables the TFFF flag in the DSPIx_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests.</p> <p>0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled</p>
TFFF_DIRS	<p>Transmit FIFO fill DMA or interrupt request select Selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPIx_SR is set, and the TFFF_RE bit in the DSPIx_RSER is set, this bit selects between generating an interrupt request or a DMA request.</p> <p>0 Interrupt request is selected 1 DMA request is selected</p>
RFOF_RE	<p>Receive FIFO overflow request enable Enables the RFOF flag in the DSPIx_SR to generate an interrupt requests.</p> <p>0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled</p>

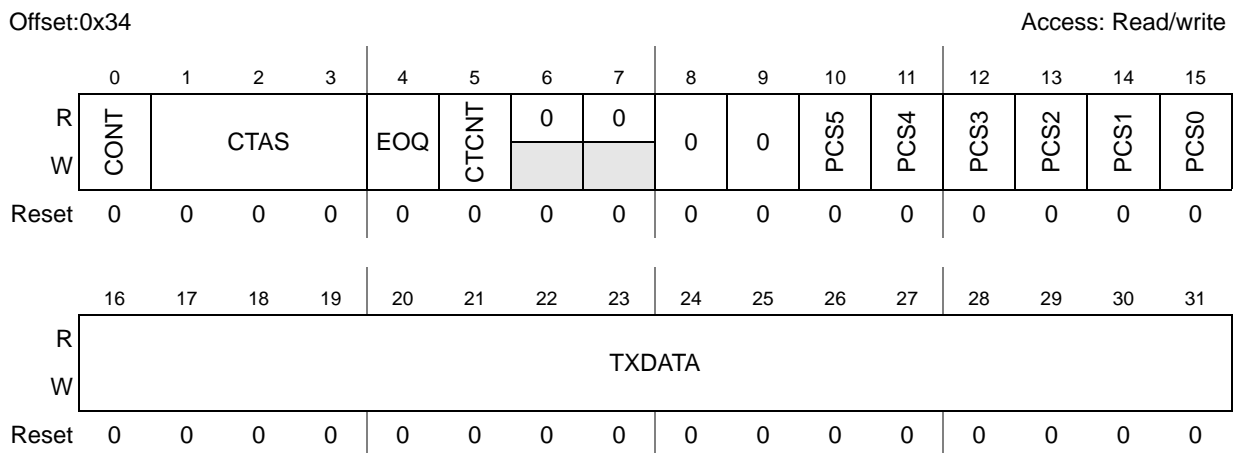
**Table 328. DSPIx\_RSER field descriptions (continued)**

Field	Description
RFDF_RE	<p>Receive FIFO drain request enable</p> <p>Enables the RFDF flag in the DSPIx_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request.</p> <p>0 RFDF interrupt requests or DMA requests are disabled                      1 RFDF interrupt requests or DMA requests are enabled</p>
RFDF_DIRS	<p>Receive FIFO drain DMA or interrupt request select</p> <p>Selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPIx_SR is set, and the RFDF_RE bit in the DSPIx_RSER is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request.</p> <p>0 Interrupt request is selected                      1 DMA request is selected</p>

**26.5.7 DSPI PUSH TX FIFO Register (DSPIx\_PUSHR)**

The DSPIx\_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See [Section 26.6.3.4: Transmit First In First Out \(TX FIFO\) buffering mechanism](#), for more information. Write accesses of 8 or 16 bits to the DSPIx\_PUSHR transfers 32 bits to the TX FIFO.

*Note:* TXDATA is used in master and slave modes.



**Figure 333. DSPI PUSH TX FIFO Register (DSPIx\_PUSHR)**



Table 329. DSPIx\_PUSHR field descriptions

Field	Description																		
CONT	<p>Continuous peripheral chip select enable                      Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected CS signals to remain asserted between transfers. See <a href="#">Section 26.6.5.5: Continuous selection format</a>, for more information.</p> <p>0 Return peripheral chip select signals to their inactive state between transfers                      1 Keep peripheral chip select signals asserted between transfers</p>																		
CTAS	<p>Clock and transfer attributes select                      Selects which of the DSPIx_CTARs is used to set the transfer attributes for the SPI frame. In SPI slave mode, DSPIx_CTAR0 is used. The following table shows how the CTAS values map to the DSPIx_CTARs. There are eight DSPIx_CTARs in the device DSPI implementation.  <b>Note:</b> Use in SPI master mode only.</p> <table border="1" data-bbox="614 779 1104 1227"> <thead> <tr> <th>CTAS</th> <th>Use clock and transfer attributes from</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>DSPIx_CTAR0</td> </tr> <tr> <td>001</td> <td>DSPIx_CTAR1</td> </tr> <tr> <td>010</td> <td>DSPIx_CTAR2</td> </tr> <tr> <td>011</td> <td>DSPIx_CTAR3</td> </tr> <tr> <td>100</td> <td>DSPIx_CTAR4</td> </tr> <tr> <td>101</td> <td>DSPIx_CTAR5</td> </tr> <tr> <td>110</td> <td>Reserved</td> </tr> <tr> <td>111</td> <td>Reserved</td> </tr> </tbody> </table>	CTAS	Use clock and transfer attributes from	000	DSPIx_CTAR0	001	DSPIx_CTAR1	010	DSPIx_CTAR2	011	DSPIx_CTAR3	100	DSPIx_CTAR4	101	DSPIx_CTAR5	110	Reserved	111	Reserved
CTAS	Use clock and transfer attributes from																		
000	DSPIx_CTAR0																		
001	DSPIx_CTAR1																		
010	DSPIx_CTAR2																		
011	DSPIx_CTAR3																		
100	DSPIx_CTAR4																		
101	DSPIx_CTAR5																		
110	Reserved																		
111	Reserved																		
EOQ	<p>End of queue                      Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPIx_SR is set.</p> <p>0 The SPI data is not the last data to transfer                      1 The SPI data is the last data to transfer  <b>Note:</b> Use in SPI master mode only.</p>																		
CTCNT	<p>Clear SPI_TCNT                      Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPIx_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins.</p> <p>0 Do not clear SPI_TCNT field in the DSPIx_TCR                      1 Clear SPI_TCNT field in the DSPIx_TCR  <b>Note:</b> Use in SPI master mode only.</p>																		

Table 329. DSPIx\_PUSHR field descriptions (continued)

Field	Description
PCSx	Peripheral chip select x Selects which CSx signals are asserted for the transfer.  0 Negate the CSx signal 1 Assert the CSx signal <b>Note:</b> Use in SPI master mode only.
TXDATA	Transmit data Holds SPI data for transfer according to the associated SPI command.  <b>Note:</b> Use TXDATA in master and slave modes.

### 26.5.8 DSPI POP RX FIFO Register (DSPIx\_POPR)

The DSPIx\_POPR allows you to read the RX FIFO. See [Section 26.6.3.5: Receive First In First Out \(RX FIFO\) buffering mechanism](#) for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPIx\_POPR fetch the RX FIFO data, and update the counter and pointer.

*Note:* Reading the RX FIFO field fetches data from the RX FIFO. Once the RX FIFO is read, the read data pointer is moved to the next entry in the RX FIFO. Therefore, read DSPIx\_POPR only when you need the data. For compatibility, configure the TLB entry for DSPIx\_POPR as guarded.

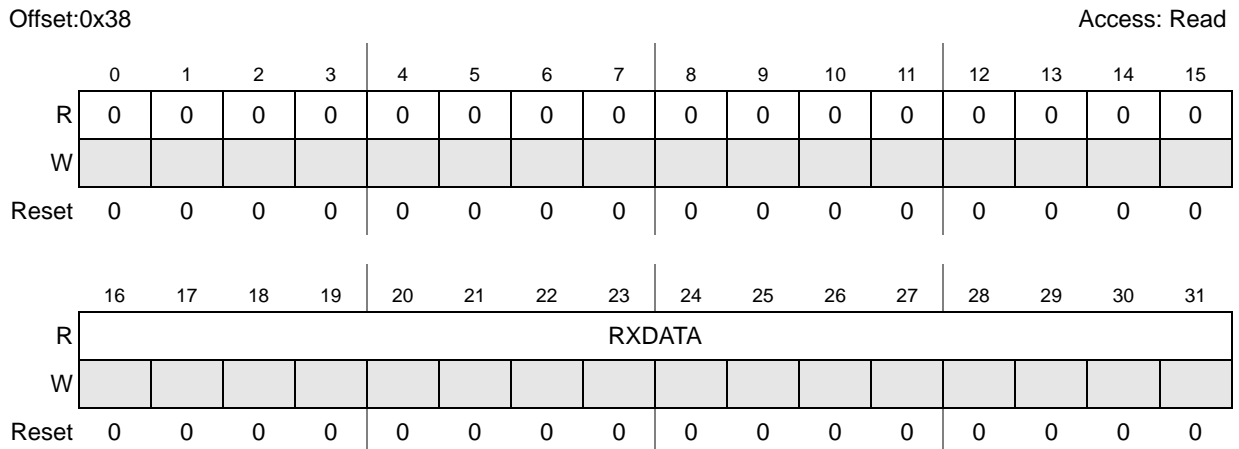


Figure 334. DSPI POP RX FIFO Register (DSPIx\_POPR)

Table 330. DSPIx\_POPR field descriptions

Field	Description
RXDATA	Received data The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNEXTPTR).

### 26.5.9 DSPI Transmit FIFO Registers 0–3 (DSPIx\_TXFRn)

The DSPIx\_TXFRn registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPIx\_TXFRn registers does not alter the state of the TX FIFO. The MCU uses four registers to implement the TX FIFO, that is DSPIx\_TXFR0–DSPIx\_TXFR3 are used.

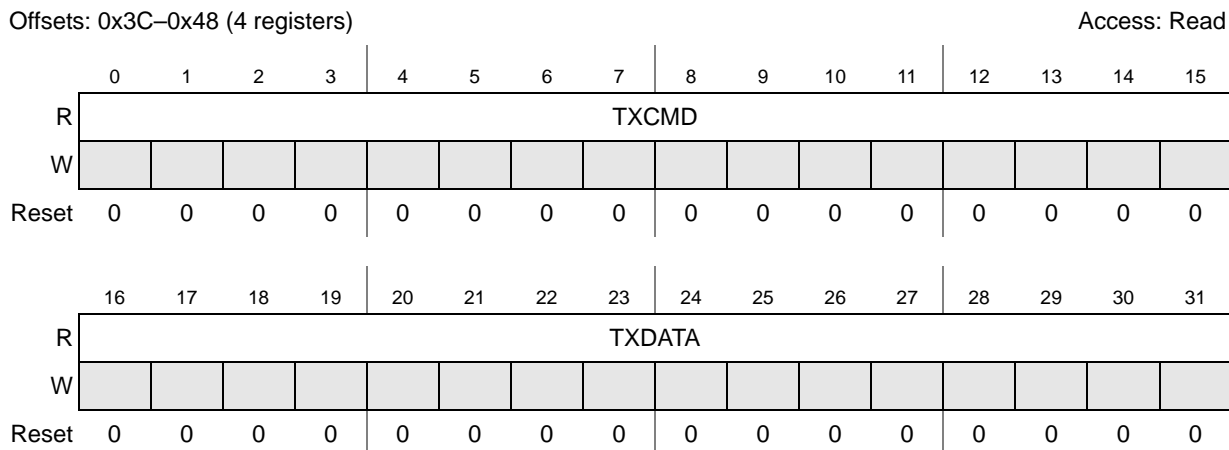


Figure 335. DSPI Transmit FIFO Register 0–3 (DSPIx\_TXFRn)

Table 331. DSPIx\_TXFRn field descriptions

Field	Description
TXCMD	Transmit command Contains the command that sets the transfer attributes for the SPI data. See <a href="#">Section 26.5.7: DSPI PUSH TX FIFO Register (DSPIx_PUSHR)</a> , for details on the command field.
TXDATA	Transmit data Contains the SPI data to be shifted out.

#### 26.5.9.1 DSPI Receive FIFO Registers 0–3 (DSPIx\_RXFRn)

The DSPIx\_RXFRn registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPIx\_RXFR registers are read-only. Reading the DSPIx\_RXFRn registers does not alter the state of the RX FIFO. The device uses four registers to implement the RX FIFO, that is DSPIx\_RXFR0–DSPIx\_RXFR3 are used.

Offsets: 0x7C–0x88 (4 registers)

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 336. DSPI Receive FIFO Registers 0–3 (DSPIx\_RXFRn)

Table 332. DSPIx\_RXFRn field description

Field	Description
RXDATA	Receive data Contains the received SPI data.

## 26.6 Functional description

The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. All communications are through an SPI-like protocol.

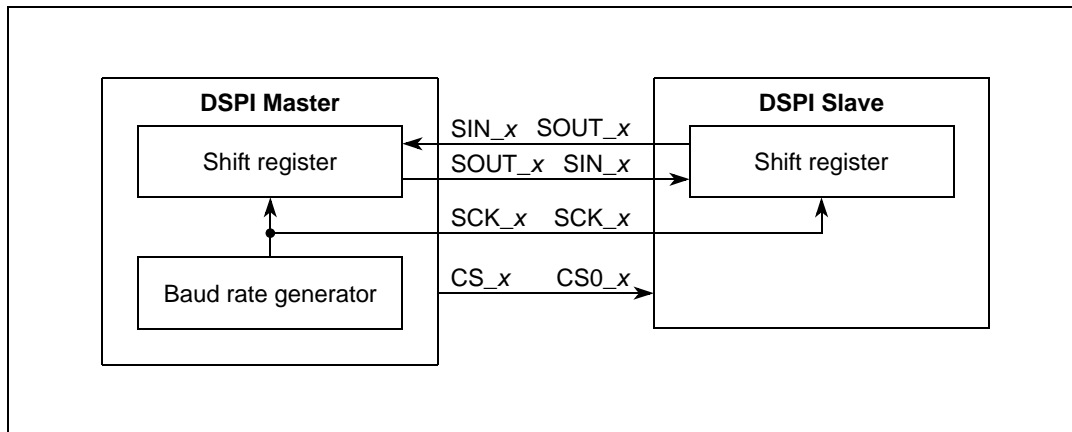
The DSPI has one configuration, namely serial peripheral interface (SPI), in which the DSPI operates as a basic SPI or a queued SPI.

The DCONF field in the DSPIx\_MCR register determines the DSPI configuration. See [Table 312](#) for the DSPI configuration values.

The DSPIx\_CTAR0–DSPIx\_CTAR5 registers hold clock and transfer attributes. The SPI configuration can select which CTAR to use on a frame by frame basis by setting the CTAS field in the DSPIx\_PUSHR.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT\_x and SIN\_x signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data is exchanged between the master and the slave; the data that was in the master's shift register is now in the shift register of the slave, and vice versa. At the end of a transfer, the TCF bit in the DSPIx\_SR is set to indicate a completed transfer. [Figure 337](#) illustrates how master and slave data is exchanged.





**Figure 337. SPI serial protocol overview**

The DSPI has six peripheral chip select (CS<sub>x</sub>) signals that are used to select which of the slaves to communicate with.

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in [Section 26.6.5: Transfer formats](#). The transfer rate and delay settings are described in [Section 26.6.4: DSPI baud rate and clock delay generation](#).

See [Section 26.6.8: Power saving features](#), for information on the power-saving features of the DSPI.

## 26.6.1 Modes of operation

The DSPI modules have the following available distinct modes:

- Master mode
- Slave mode
- Module Disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes whereas debug mode device-specific.

The module-specific modes are determined by bits in the DSPI<sub>x</sub>\_MCR. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

### 26.6.1.1 Master mode

In master mode the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the DSPI<sub>x</sub>\_MCR is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI<sub>x</sub>\_CTARs are used to set the transfer attributes. Transfer attribute control is on a frame by frame basis.

See [Section 26.6.3: Serial peripheral interface \(SPI\) configuration](#) for more details.

### 26.6.1.2 Slave mode

In slave mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPIx\_MCR is negated. The DSPI slave is selected by a bus master by having the slave's CS0\_x asserted. In slave mode the SCK is provided by the bus master. All transfer attributes are controlled by the bus master, except the clock polarity, clock phase and the number of bits to transfer which must be configured in the DSPI slave to communicate correctly.

### 26.6.1.3 Module Disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPIx\_MCR is set.

See [Section 26.6.8: Power saving features](#), for more details on the module disable mode.

### 26.6.1.4 Debug mode

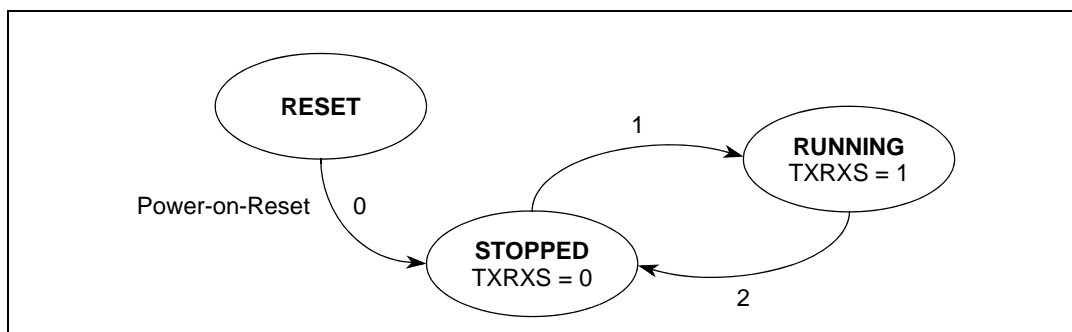
The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPIx\_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller.

See [Figure 338](#) for a state diagram.

## 26.6.2 Start and stop of DSPI transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPIx\_SR is cleared in this state. In the RUNNING state, serial transfers take place. The TXRXS bit in the DSPIx\_SR is set in the RUNNING state.

[Figure 338](#) shows a state diagram of the start and stop mechanism.



**Figure 338. DSPI start and stop state diagram**

The transitions are described in [Table 333](#).

**Table 333. State transitions for start and stop of DSPI transfers**

Transition No.	Current state	Next state	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI starts (transitions from STOPPED to RUNNING) when all of the following conditions are true: <ul style="list-style-type: none"> <li>– EOQF bit is clear</li> <li>– Debug mode is unselected or the FRZ bit is clear</li> <li>– HALT bit is clear</li> </ul>
2	RUNNING	STOPPED	The DSPI stops (transitions from RUNNING to STOPPED) after the current frame for any one of the following conditions: <ul style="list-style-type: none"> <li>– EOQF bit is set</li> <li>– Debug mode is selected and the FRZ bit is set</li> <li>– HALT bit is set</li> </ul>

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

### 26.6.3 Serial peripheral interface (SPI) configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when the DCONF field in the DSPIx\_MCR is 0b00. The SPI frames can be from 4 to 16 bits long. The data to be transmitted can come from queues stored in SRAM external to the DSPI. Host software or an eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or an eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI.

The FIFO buffer operations are described in [Section 26.6.3.4: Transmit First In First Out \(TX FIFO\) buffering mechanism](#), and [Section 26.6.3.5: Receive First In First Out \(RX FIFO\) buffering mechanism](#).

The interrupt and DMA request conditions are described in [Section 26.6.7: Interrupt/DMA requests](#).

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is ignored.

#### 26.6.3.1 SPI Master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the serial communications clock (SCK\_x) and the peripheral chip select (CSx) signals. The SPI

command field in the executing TX FIFO entry determines which CTARs are used to set the transfer attributes and which CSx signal to assert. The command field also contains various bits that help with queue management and transfer protocol. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (SOUT\_x) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

See [Section 26.5.7: DSPI PUSH TX FIFO Register \(DSPIx\\_PUSHR\)](#), for details on the SPI command fields.

### 26.6.3.2 SPI Slave mode

In SPI slave mode the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with an SPI master. The SPI slave mode transfer attributes are set in the DSPIx\_CTAR0.

### 26.6.3.3 FIFO disable operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a '1' to the DIS\_TXF bit in the DSPIx\_MCR. The RX FIFO is disabled by writing a '1' to the DIS\_RXF bit in the DSPIx\_MCR.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPIx\_PUSHR and received data is read from the DSPIx\_POPR. When the TX FIFO is disabled, the TFFF, TFUF, and TXCTR fields in DSPIx\_SR behave as if there is a one-entry FIFO but the contents of the DSPIx\_TXFRs and TXNXTPTR are undefined. When the RX FIFO is disabled, the RFDF, RFOF, and RXCTR fields in the DSPIx\_SR behave as if there is a one-entry FIFO but the contents of the DSPIx\_RXFRs and POPNXTPTR are undefined.

Disable the TX and RX FIFOs only if the FIFO must be disabled as a requirement of the application's operating mode. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported, and can result in incorrect results.

### 26.6.3.4 Transmit First In First Out (TX FIFO) buffering mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds four entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPIx\_PUSHR). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO. For more information on DSPIx\_PUSHR, see [Section 26.5.7: DSPI PUSH TX FIFO Register \(DSPIx\\_PUSHR\)](#).

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPIx\_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI\_PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

See [Section 26.5.5: DSPI Status Register \(DSPIx\\_SR\)](#) for more information on DSPIx\_SR.

The TXNXTPTR field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPIx\_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPIx\_TXFR2 contains

the SPI data and command for the next transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register.

#### 26.6.3.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPIx\_PUSHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPIx\_SR is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPIx\_PUSHR is complete or alternatively by host software writing a '1' to the TFFF in the DSPIx\_SR. The TFFF can generate a DMA request or an interrupt request.

See [Section 26.6.7.2: Transmit FIFO Fill Interrupt or DMA Request \(TFFF\)](#), for details.

The DSPI ignores attempts to push data to a full TX FIFO; that is, the state of the TX FIFO is unchanged. No error condition is indicated.

#### 26.6.3.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter is decremented by one. At the end of a transfer, the TCF bit in the DSPIx\_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a '1' to the CLR\_TXF bit in DSPIx\_MCR.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave's DSPI TX FIFO is empty, the transmit FIFO underflow flag (TFUF) in the slave's DSPIx\_SR is set.

See [Section 26.6.7.4: Transmit FIFO Underflow Interrupt Request \(TFUF\)](#), for details.

#### 26.6.3.5 Receive First In First Out (RX FIFO) buffering mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds four received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is removed (popped) from the RX FIFO by reading the DSPIx\_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPIx\_POPR or by flushing the RX FIFO.

See [Section 26.5.8: DSPI POP RX FIFO Register \(DSPIx\\_POPR\)](#) for more information on the DSPIx\_POPR.

The RX FIFO counter field (RXCTR) in the DSPI status register (DSPIx\_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPIx\_POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPIx\_SR points to the RX FIFO entry that is returned when the DSPIx\_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPIx\_RXFR0. For example, POPNXTPTR equal to two means that the DSPIx\_RXFR2 contains the received SPI data that is returned when DSPIx\_POPR is read. The POPNXTPTR field is incremented every time the DSPIx\_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

### 26.6.3.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO the RX FIFO counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPIx\_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPIx\_MCR, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored.

### 26.6.3.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPIx\_POPR. A read of the DSPIx\_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

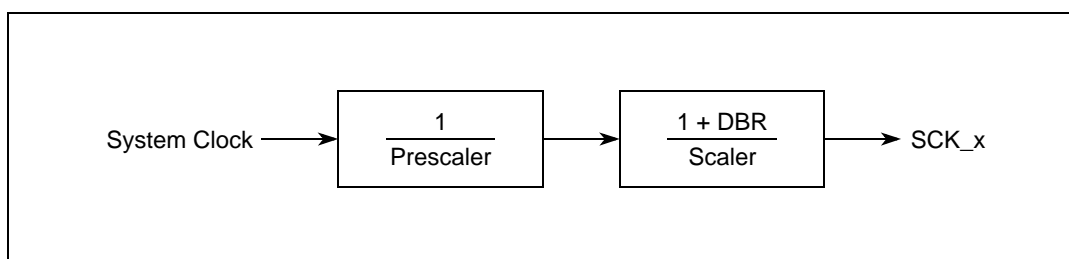
See [Section 26.5.8: DSPI POP RX FIFO Register \(DSPIx\\_POPR\)](#) for more information on DSPIx\_POPR.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPIx\_SR is set. The RFDF bit is cleared when the RX\_FIFO is empty and the eDMA controller indicates that a read from DSPIx\_POPR is complete; alternatively the RFDF bit can be cleared by the host writing a '1' to it.

## 26.6.4 DSPI baud rate and clock delay generation

The SCK\_x frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate.

[Figure 339](#) shows conceptually how the SCK signal is generated.



**Figure 339. Communications clock prescalers and scalers**

### 26.6.4.1 Baud rate generator

The baud rate is the frequency of the serial communication clock (SCK\_x). The system clock is divided by a baud rate prescaler (defined by DSPIx\_CTAR[PBR]) and baud rate scaler (defined by DSPIx\_CTAR[BR]) to produce SCK\_x with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPIx\_CTARs select the frequency of SCK\_x using the following formula:

$$\text{SCK baud rate} = \frac{f_{\text{SYS}}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 334 shows an example of a computed baud rate.

**Table 334. Baud rate computation example**

f <sub>sys</sub>	PBR	Prescaler value	BR	Scaler value	DBR value	Baud rate
64 MHz	0b00	2	0b0000	2	0	16 Mbit/s
20 MHz	0b00	2	0b0000	2	1	10 Mbit/s

### 26.6.4.2 CS to SCK delay (t<sub>csc</sub>)

The CS<sub>x</sub> to SCK<sub>x</sub> delay is the length of time from assertion of the CS<sub>x</sub> signal to the first SCK<sub>x</sub> edge. See Figure 341 for an illustration of the CS<sub>x</sub> to SCK<sub>x</sub> delay. The PCSSCK and CSSCK fields in the DSPI<sub>x</sub>\_CTAR<sub>n</sub> registers select the CS<sub>x</sub> to SCK<sub>x</sub> delay, and the relationship is expressed by the following formula:

$$t_{\text{csc}} = \frac{1}{f_{\text{SYS}}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 335 shows an example of the computed CS to SCK<sub>x</sub> delay.

**Table 335. CS to SCK delay computation example**

PCSSCK	Prescaler value	CSSCK	Scaler value	f <sub>sys</sub>	CS to SCK delay
0b01	3	0b0100	32	64 MHz	1.5 μs

### 26.6.4.3 After SCK delay (t<sub>asc</sub>)

The after SCK<sub>x</sub> delay is the length of time between the last edge of SCK<sub>x</sub> and the negation of CS<sub>x</sub>. See Figure 341 and Figure 342 for illustrations of the after SCK<sub>x</sub> delay. The PASC and ASC fields in the DSPI<sub>x</sub>\_CTAR<sub>n</sub> registers select the after SCK delay. The relationship between these variables is given in the following formula:

$$t_{\text{asc}} = \frac{1}{f_{\text{SYS}}} \times \text{PASC} \times \text{ASC}$$

Table 336 shows an example of the computed after SCK delay.

**Table 336. After SCK delay computation example**

PASC	Prescaler value	ASC	Scaler value	f <sub>sys</sub>	After SCK delay
0b01	3	0b0100	32	64 MHz	1.5 μs

**26.6.4.4 Delay after transfer (t<sub>DT</sub>)**

The delay after transfer is the length of time between negation of the CS<sub>x</sub> signal for a frame and the assertion of the CS<sub>x</sub> signal for the next frame. The PDT and DT fields in the DSPI<sub>x</sub>\_CTAR<sub>n</sub> registers select the delay after transfer.

See [Figure 341](#) for an illustration of the delay after transfer.

The following formula expresses the PDT/DT/delay after transfer relationship:

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

[Table 337](#) shows an example of the computed delay after transfer.

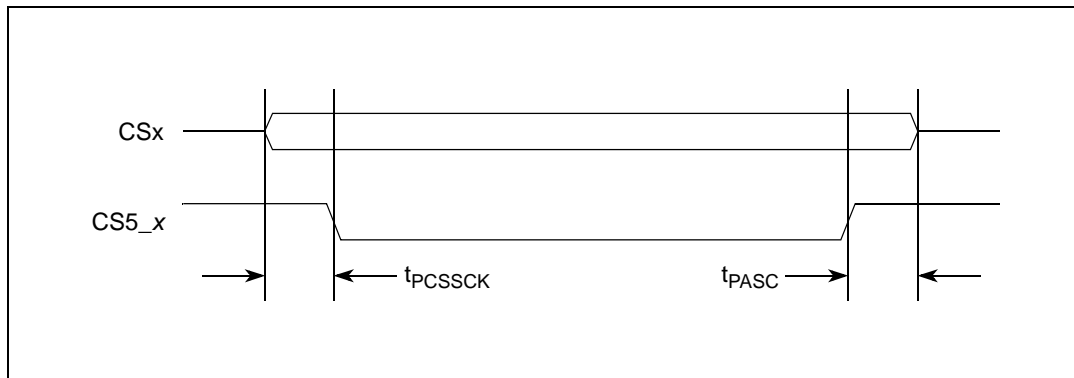
**Table 337. Delay after transfer computation example**

PDT	Prescaler value	DT	Scaler value	f <sub>SYS</sub>	Delay after transfer
0b01	3	0b1110	32768	64 MHz	1.54 ms

**26.6.4.5 Peripheral chip select strobe enable (CS5\_x)**

The CS5\_x signal provides a delay to allow the CS<sub>x</sub> signals to settle after transitioning thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI<sub>x</sub>\_MCR, CS5\_x provides a signal for an external demultiplexer to decode the CS4\_x signals into as many as 32 glitch-free CS<sub>x</sub> signals.

[Figure 340](#) shows the timing of the CS5\_x signal relative to CS signals.



**Figure 340. Peripheral chip select strobe timing**

The delay between the assertion of the CS<sub>x</sub> signals and the assertion of CS5\_x is selected by the PCSSCK field in the DSPI<sub>x</sub>\_CTAR based on the following formula:

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between CS5\_x negation and CS<sub>x</sub> negation is selected by the PASC field in the DSPI<sub>x</sub>\_CTAR based on the following formula:



$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 338 shows an example of the computed  $t_{PCSSCK}$  delay.

**Table 338. Peripheral chip select strobe assert computation example**

PCSSCK	Prescaler	$f_{SYS}$	Delay before transfer
0b11	7	64 MHz	109.4 ns

Table 339 shows an example of the computed the  $t_{PASC}$  delay.

**Table 339. Peripheral chip select strobe negate computation example**

PASC	Prescaler	$f_{SYS}$	Delay after transfer
0b11	7	64 MHz	109.4 ns

### 26.6.5 Transfer formats

The SPI serial communication is controlled by the serial communications clock (SCK\_x) signal and the CSx signals. The SCK\_x signal provided by the master device synchronizes shifting and sampling of the data by the SIN\_x and SOUT\_x pins. The CSx signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI clock and transfer attributes registers (DSPIx\_CTARn) select the polarity and phase of the serial clock, SCK\_x. The polarity bit selects the idle state of the SCK\_x. The clock phase bit selects if the data on SOUT\_x is valid before or on the first SCK\_x edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPIx\_CTAR0 (SPI slave mode) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPIx\_MCR selects between classic SPI format and modified transfer format. The classic SPI formats are described in [Section 26.6.5.1: Classic SPI transfer format \(CPHA = 0\)](#) and [Section 26.6.5.2: Classic SPI transfer format \(CPHA = 1\)](#). The modified transfer formats are described in [Section 26.6.5.3: Modified SPI transfer format \(MTFE = 1, CPHA = 0\)](#) and [Section 26.6.5.4: Modified SPI transfer format \(MTFE = 1, CPHA = 1\)](#).

In the SPI configuration, the DSPI provides the option of keeping the CS signals asserted between frames. See [Section 26.6.5.5: Continuous selection format](#) for details.

26.6.5.1 Classic SPI transfer format (CPHA = 0)

The transfer format shown in [Figure 341](#) is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN<sub>x</sub> pins on the odd-numbered SCK<sub>x</sub> edges and change the data on their SOUT<sub>x</sub> pins on the even-numbered SCK<sub>x</sub> edges.

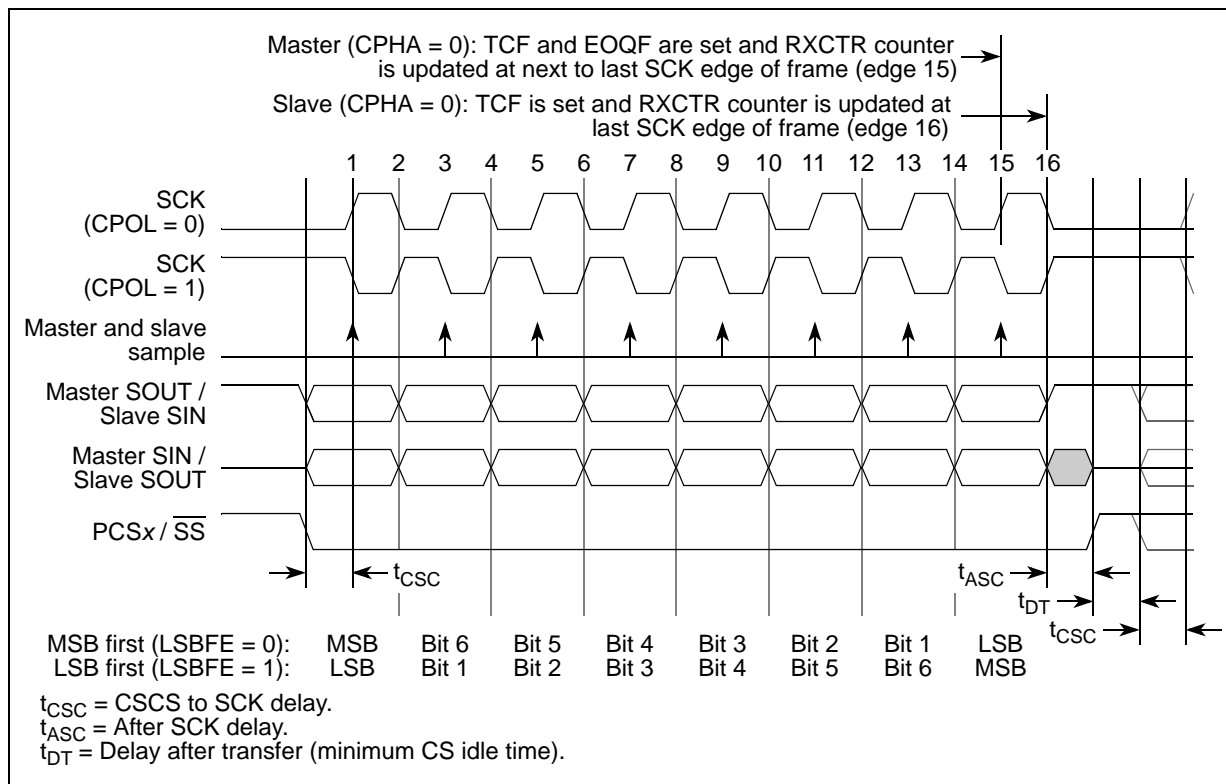


Figure 341. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the SOUT<sub>x</sub> pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT<sub>x</sub> pin. After the  $t_{CSC}$  delay has elapsed, the master outputs the first edge of SCK<sub>x</sub>. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK<sub>x</sub> the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN<sub>x</sub> pins on the odd-numbered clock edges and changes the data on their SOUT<sub>x</sub> pins on the even-numbered clock edges. After the last clock edge occurs a delay of  $t_{ASC}$  is inserted before the master negates the CS signals. A delay of  $t_{DT}$  is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of [Figure 341](#).

For the CPHA = 0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of [Figure 341](#).

26.6.5.2 Classic SPI transfer format (CPHA = 1)

This transfer format shown in [Figure 342](#) is used to communicate with peripheral SPI slave devices that require the first SCK\_x edge before the first data bit becomes available on the slave SOUT\_x pin. In this format the master and slave devices change the data on their SOUT\_x pins on the odd-numbered SCK\_x edges and sample the data on their SIN\_x pins on the even-numbered SCK\_x edges.

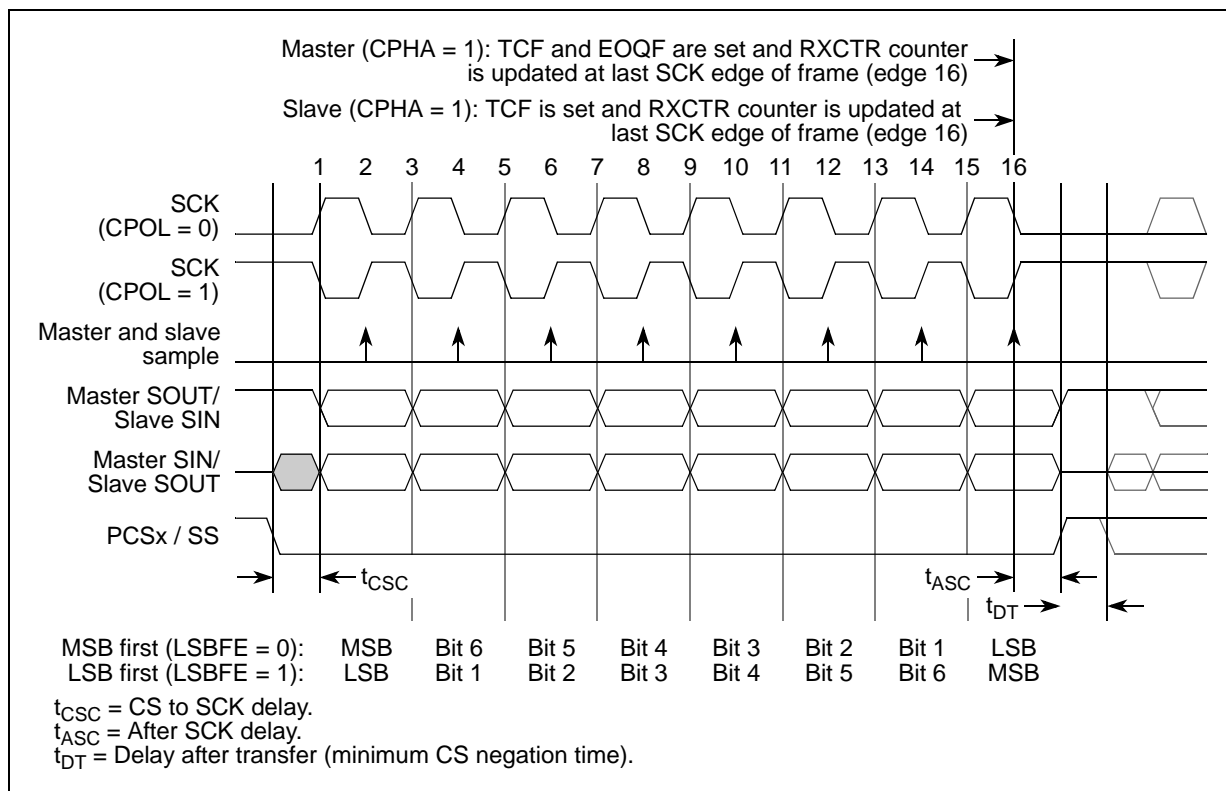


Figure 342. DSPI transfer timing diagram (MFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the CSx signal to the slave. After the  $t_{CSC}$  delay has elapsed, the master generates the first SCK\_x edge and at the same time places valid data on the master SOUT\_x pin. The slave responds to the first SCK\_x edge by placing its first data bit on its slave SOUT\_x pin.

At the second edge of the SCK\_x the master and slave sample their SIN\_x pins. For the rest of the frame the master and the slave change the data on their SOUT\_x pins on the odd-numbered clock edges and sample their SIN\_x pins on the even-numbered clock edges. After the last clock edge occurs a delay of  $t_{ASC}$  is inserted before the master negates the CSx signal. A delay of  $t_{DT}$  is inserted before a new frame transfer can be initiated by the master.

For CPHA = 1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of [Figure 342](#). For CPHA = 1 the master and slave RXCTR counters are updated on the same clock edge.

**26.6.5.3 Modified SPI transfer format (MTFE = 1, CPHA = 0)**

In this modified transfer format both the master and the slave sample later in the SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

*Note:* For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

The master and the slave place data on the SOUT\_x pins at the assertion of the CSx signal. After the CSx to SCK\_x delay has elapsed the first SCK\_x edge is generated. The slave samples the master SOUT\_x signal on every odd numbered SCK\_x edge. The slave also places new data on the slave SOUT\_x on every odd numbered clock edge.

The master places its second data bit on the SOUT\_x line one system clock after odd numbered SCK\_x edge. The point where the master samples the slave SOUT\_x is selected by writing to the SMPL\_PT field in the DSPIx\_MCR. Table 340 lists the number of system clock cycles between the active edge of SCK\_x and the master sample point for different values of the SMPL\_PT bit field. The master sample point can be delayed by one or two system clock cycles.

**Table 340. Delayed master sample point**

SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK and sampling of SIN
00	0
01	1
10	2
11	Invalid value

Figure 343 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

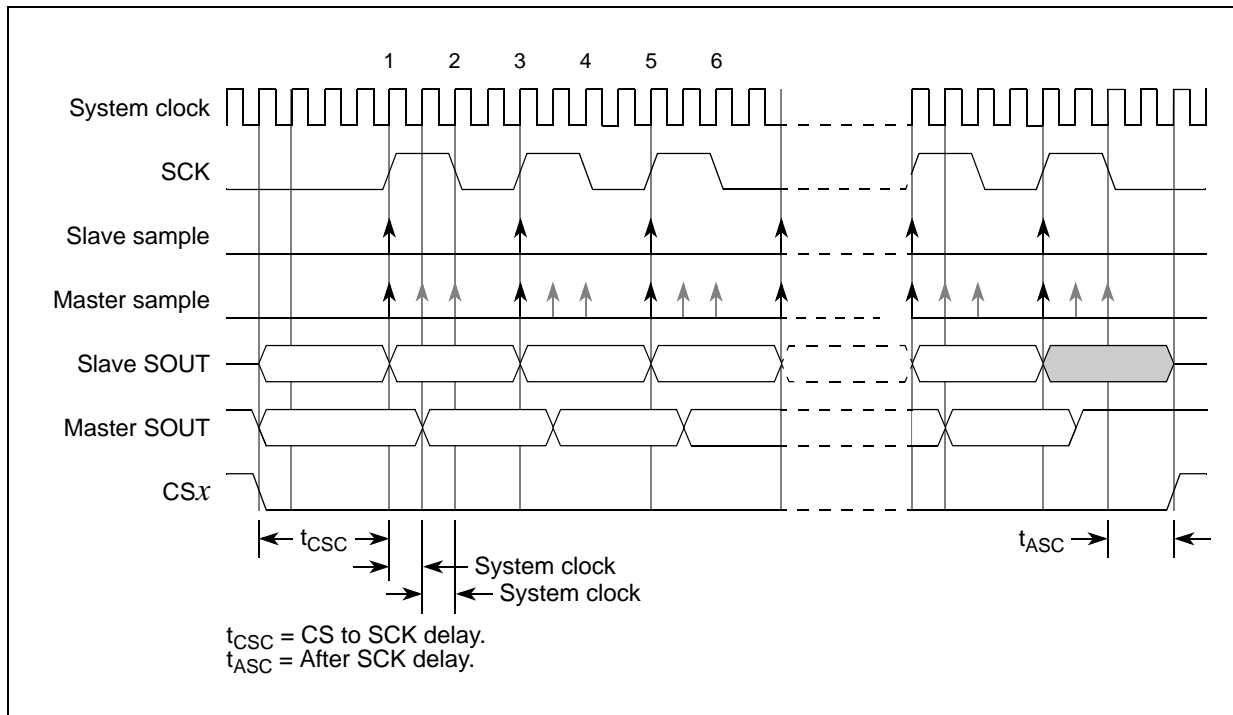


Figure 343. DSPI modified transfer format (MTFE = 1, CPHA = 0,  $f_{SCK} = f_{SYS} / 4$ )

#### 26.6.5.4 Modified SPI transfer format (MTFE = 1, CPHA = 1)

At the start of a transfer the DSPI asserts the CS signal to the slave device. After the CS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the third SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge is visible on the master SCK pin during the sampling of the last bit. The SCK to CS delay must be greater or equal to half of the SCK period.

*Note:* For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

Figure 344 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is described.

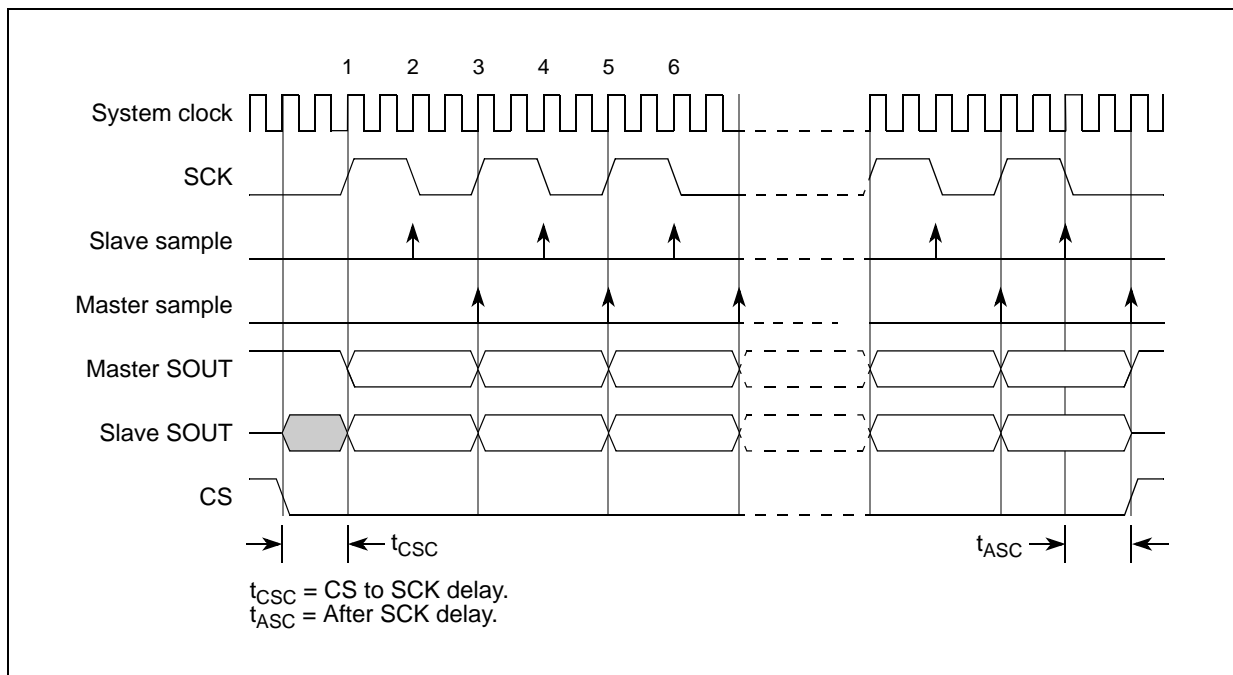


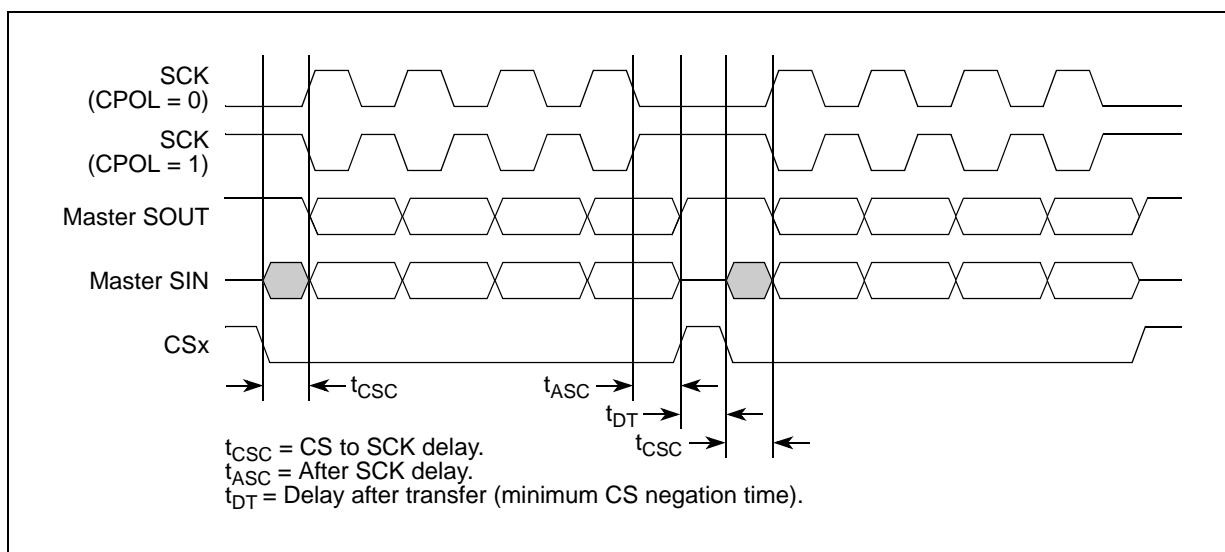
Figure 344. DSPI modified transfer format (MTFE = 1, CPHA = 1,  $f_{SCK} = f_{sys} / 4$ )

### 26.6.5.5 Continuous selection format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the CONT bit in the SPI command.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS field in the DSPIx\_MCR.

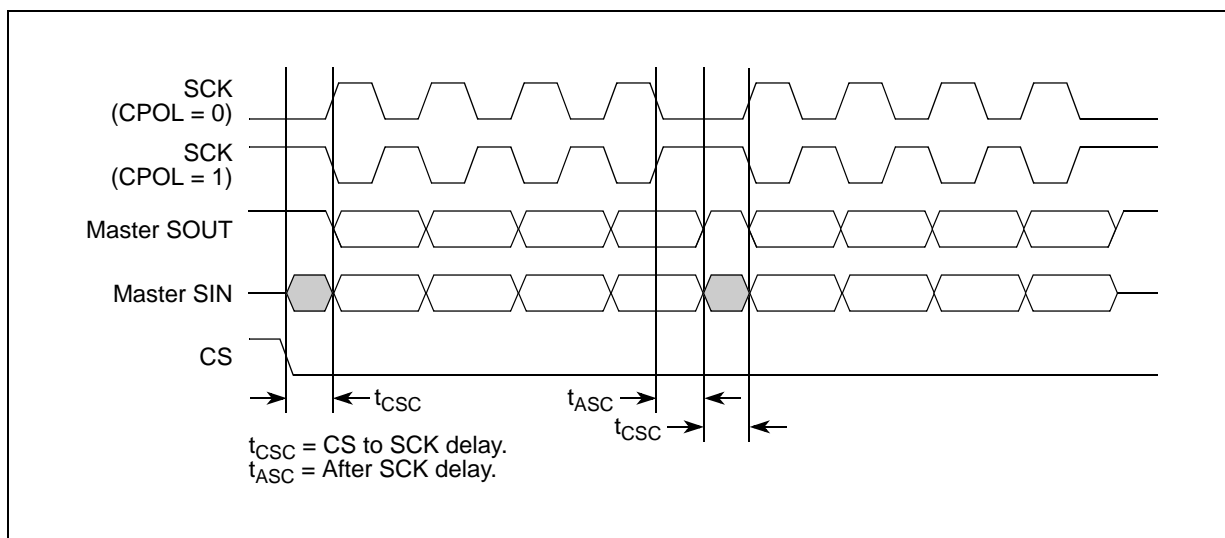
Figure 345 shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.



**Figure 345. Example of non-continuous format (CPHA = 1, CONT = 0)**

When the CONT = 1 and the CS signal for the next transfer is the same as for the current transfer, the CS signal remains asserted for the duration of the two transfers. The delay between transfers ( $t_{DT}$ ) is not inserted between the transfers.

*Figure 346* shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.



**Figure 346. Example of continuous transfer (CPHA = 1, CONT = 1)**

In *Figure 346*, the period length at the start of the next transfer is the sum of  $t_{ASC}$  and  $t_{CSC}$ ; that is, it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations,  $t_{ASC}$  and  $t_{CSC}$  must be increased if a full half-clock period is required.

Switching CTARs between frames while using continuous selection can cause errors in the transfer. The CS signal must be negated before CTAR is switched.

When the CONT bit = 1 and the CS signals for the next transfer are different from the present transfer, the CS signals behave as if the CONT bit was not set.

*Note:* You must fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example; while transmitting in master mode, ensure that the last entry in the TXFIFO, after which TXFIFO becomes empty, has CONT = 0 in the command frame.

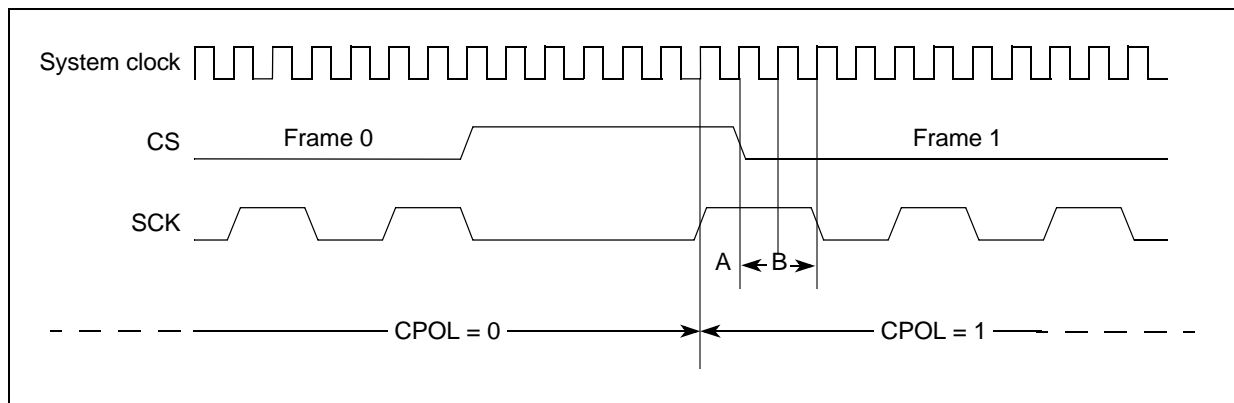
When operating in slave mode, ensure that when the last-entry in the TXFIFO is completely transmitted (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave is deselected for any further serial communication; otherwise, an underflow error occurs.

**26.6.5.6 Clock polarity switching between DSPI transfers**

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame.

See [Section 26.5.4: DSPI Clock and Transfer Attributes Registers 0–5 \(DSPIx\\_CTARn\)](#).

In [Figure 347](#), time ‘A’ shows the one clock interval. Time ‘B’ is user programmable from a minimum of two system clocks.



**Figure 347. Polarity switching between frames**

**26.6.6 Continuous serial communications clock**

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT\_SCKE bit in the DSPIx\_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 is ignored if the CONT\_SCKE bit is set. Continuous SCK is supported for modified transfer format.



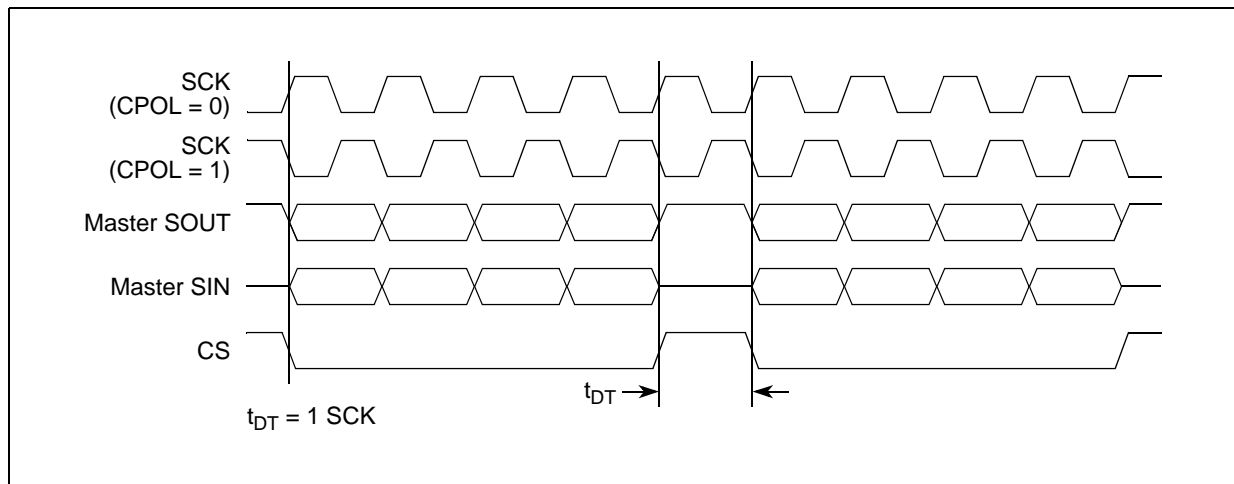
Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- The TX FIFO must be cleared before initiating any SPI configuration transfer.
- When the DSPI is in SPI configuration, CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame should be CTAR0.
- In all configurations, the currently selected CTAR remains in use until the start of a frame with a different CTAR specified, or the continuous SCK mode is terminated.

The device is designed to use the same baud rate for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the CS to SCK delay and the After SCK delay. The delay after transfer is fixed at one SCK cycle. *Figure 348* shows timing diagram for continuous SCK format with continuous selection disabled.

*Note:* When in Continuous SCK mode, always use CTAR0 for the SPI transfer, and clear the TXFIFO using the MCR[CLR\_TXF] field before initiating transfer.



**Figure 348. Continuous SCK timing diagram (CONT= 0)**

If the CONT bit in the TX FIFO entry is set, CS remains asserted between the transfers when the CS signal for the next transfer is the same as for the current transfer. *Figure 349* shows timing diagram for continuous SCK format with continuous selection enabled.

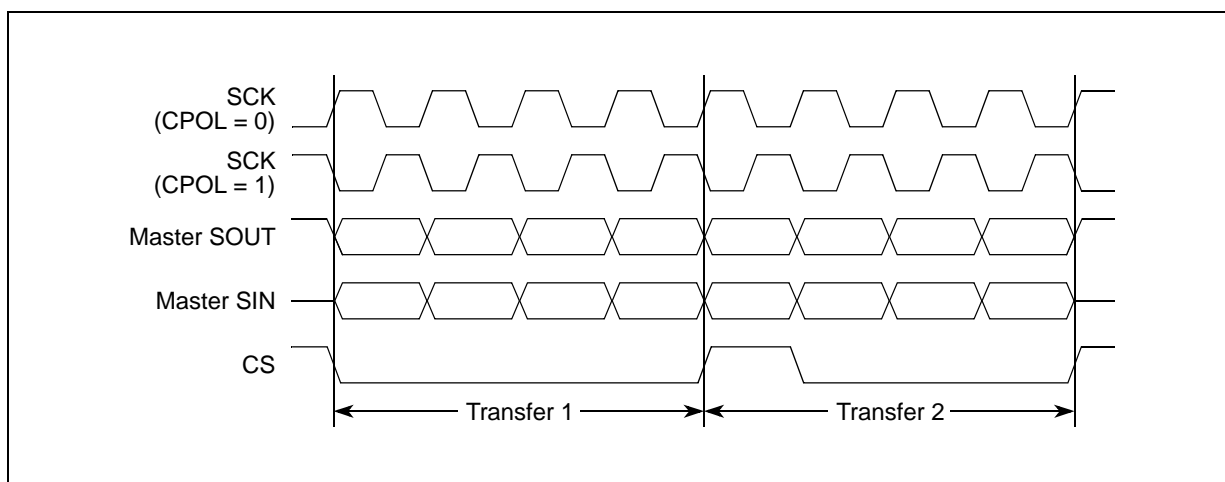


Figure 349. Continuous SCK timing diagram (CONT=1)

### 26.6.7 Interrupt/DMA requests

The DSPI has five conditions that can generate interrupt requests only, and two conditions that can generate interrupt or DMA requests.

Table 341 lists the seven conditions.

Table 341. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	
TX FIFO underflow has occurred	TFUF	X	
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow occurred	RFOF	X	
A FIFO overrun occurred <sup>(1)</sup>	TFUF ORed with RFOF	X	

1. The FIFO overrun condition is created by ORing the TFUF and RFOF flags together.

Each condition has a flag bit and a request enable bit. The flag bits are described in the [Section 26.5.5: DSPI Status Register \(DSPIx\\_SR\)](#) and the request enable bits are described in the [Section 26.5.6: DSPI DMA / Interrupt Request Select and Enable Register \(DSPIx\\_RSER\)](#). The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the TFFF\_DIRS and RFDF\_DIRS bits in the DSPIx\_RSER.

#### 26.6.7.1 End of Queue Interrupt Request (EOQF)

The end of queue request indicates that the end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF\_RE bit in the DSPIx\_RSER is set. See the EOQ bit description in

*Section 26.5.5: DSPI Status Register (DSPIx\_SR)*. See *Figure 341* and *Figure 342* that illustrate when EOQF is set.

#### 26.6.7.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates that the TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF\_RE bit in the DSPIx\_RSER is set. The TFFF\_DIRS bit in the DSPIx\_RSER selects whether a DMA request or an interrupt request is generated.

#### 26.6.7.3 Transfer Complete Interrupt Request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the TCF\_RE bit is set in the DSPIx\_RSER. See the TCF bit description in *Section 26.5.5: DSPI Status Register (DSPIx\_SR)*. See *Figure 341* and *Figure 342* that illustrate when TCF is set.

#### 26.6.7.4 Transmit FIFO Underflow Interrupt Request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode and SPI configuration is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUF\_RE bit in the DSPIx\_RSER is set, an interrupt request is generated.

#### 26.6.7.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the RFDF\_RE bit in the DSPIx\_RSER is set. The RFDF\_DIRS bit in the DSPIx\_RSER selects whether a DMA request or an interrupt request is generated.

#### 26.6.7.6 Receive FIFO Overflow Interrupt Request (RFOF)

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOF\_RE bit in the DSPIx\_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPIx\_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

#### 26.6.7.7 FIFO Overrun Request (TFUF) or (RFOF)

The FIFO overrun request indicates that at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR'ing together the RX FIFO overflow and TX FIFO underflow signals.

## 26.6.8 Power saving features

The DSPI supports the following power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

### 26.6.8.1 Module Disable mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a '1' to the MDIS bit in the DSPIx\_MCR. In module disable mode, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs does not have any effect in the module disable mode. Changes to the DIS\_TXF and DIS\_RXF fields of the DSPIx\_MCR does not have any affect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no affect. Writing to the DSPIx\_TCR during module disable mode does not have an effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

### 26.6.8.2 Slave interface signal gating

The DSPI module enable signal is used to gate slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

## 26.7 Initialization and application information

### 26.7.1 How to change queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to change queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag (EOQF) in the DSPIx\_SR is set.
3. The setting of the EOQF flag disables both serial transmission, and serial reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The eDMA continues to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPIx\_SR or by checking RFDF in the DSPIx\_SR after each read operation of the DSPIx\_POPR.

7. Modify DMA descriptor of TX and RX channels for “new” queues.
8. Flush TX FIFO by writing a ‘1’ to the CLR\_TXF bit in the DSPIx\_MCR register and flush the RX FIFO by writing a ‘1’ to the CLR\_RXF bit in the DSPIx\_MCR register.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI\_TCNT field in the DSPIx\_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

### 26.7.2 Baud rate settings

Table 342 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPIx\_CTARs. The values are calculated at a 64 MHz system frequency.

**Table 342. Baud rate values**

		Baud rate divider prescaler values (DSPI_CTAR[PBR])			
		2	3	5	7
Baud rate scaler values (DSPI_CTAR[BR])	2	16.0 MHz	10.7 MHz	6.4 MHz	4.57 MHz
	4	8 MHz	5.33 MHz	3.2 MHz	2.28 MHz
	6	5.33 MHz	3.56 MHz	2.13 MHz	1.52 MHz
	8	4 MHz	2.67 MHz	1.60 MHz	1.15 MHz
	16	2 MHz	1.33 MHz	800 kHz	571 kHz
	32	1 MHz	670 kHz	400 kHz	285 kHz
	64	500 kHz	333 kHz	200 kHz	142 kHz
	128	250 kHz	166 kHz	100 kHz	71.7 kHz
	256	125 kHz	83.2 kHz	50 kHz	35.71 kHz
	512	62.5 kHz	41.6 kHz	25 kHz	17.86 kHz
	1024	31.2 kHz	20.8 kHz	12.5 kHz	8.96 kHz
	2048	15.6 kHz	10.4 kHz	6.25 kHz	4.47 kHz
	4096	7.81 kHz	5.21 kHz	3.12 kHz	2.23 kHz
	8192	3.90 kHz	2.60 kHz	1.56 kHz	1.11 kHz
	16384	1.95 kHz	1.31 kHz	781 Hz	558 Hz
32768	979 Hz	653 Hz	390 Hz	279 Hz	

### 26.7.3 Delay settings

*Table 343* shows the values for the delay after transfer ( $t_{DT}$ ) that can be generated based on the prescaler values and the scaler values set in the DSPIx\_CTARs. The values calculated assume a 64 MHz system frequency.

**Table 343. Delay values**

		Delay prescaler values (DSPI_CTAR[PDT])			
		1	3	5	7
Delay scaler values (DSPI_CTAR[DT])	2	31.25 ns	93.75 ns	156.25 ns	218.75 ns
	4	62.5 ns	187.5 ns	312.5 ns	437.5 ns
	8	125 ns	375 ns	625 ns	875 ns
	16	250 ns	750 ns	1.25 $\mu$ s	1.75 $\mu$ s
	32	0.5 $\mu$ s	1.5 $\mu$ s	2.5 $\mu$ s	3.5 $\mu$ s
	64	1 $\mu$ s	3 $\mu$ s	5 $\mu$ s	7 $\mu$ s
	128	2 $\mu$ s	6 $\mu$ s	10 $\mu$ s	14 $\mu$ s
	256	4 $\mu$ s	12 $\mu$ s	20 $\mu$ s	28 $\mu$ s
	512	8 $\mu$ s	24 $\mu$ s	40 $\mu$ s	56 $\mu$ s
	1024	16 $\mu$ s	48 $\mu$ s	80 $\mu$ s	112 $\mu$ s
	2048	32 $\mu$ s	96 $\mu$ s	160 $\mu$ s	224 $\mu$ s
	4096	64 $\mu$ s	192 $\mu$ s	320 $\mu$ s	448 $\mu$ s
	8192	128 $\mu$ s	384 $\mu$ s	640 $\mu$ s	896 $\mu$ s
	16384	256 $\mu$ s	768 $\mu$ s	1.28 ms	1.79 ms
	32768	512 $\mu$ s	1.54 ms	2.56 ms	3.58 ms
65536	1.02 ms	3.07 ms	5.12 ms	7.17 ms	

### 26.7.4 Calculation of FIFO pointer addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR).

See [Section 26.6.3.4: Transmit First In First Out \(TX FIFO\) buffering mechanism](#), and [Section 26.6.3.5: Receive First In First Out \(RX FIFO\) buffering mechanism](#), for details on the FIFO operation. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO.

Figure 350 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter.

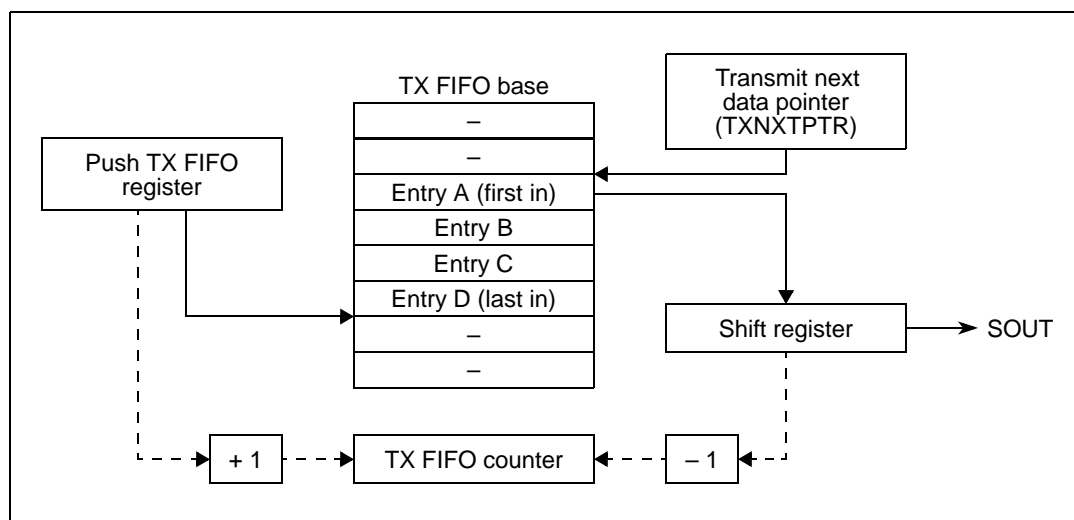


Figure 350. TX FIFO pointers and counter

**26.7.4.1 Address calculation for the first-in entry and last-in entry in the TX FIFO**

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{TXFIFO base} + 4 (\text{TXNXPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{TXFIFO base} + 4 \times [(\text{TXCTR} + \text{TXNXPTR} - 1) \text{ modulo TXFIFO depth}]$$

where:

TXFIFO base = base address of transmit FIFO

TXCTR = transmit FIFO counter

TXNXPTR = transmit next pointer

TX FIFO depth = transmit FIFO depth, implementation specific

**26.7.4.2 Address calculation for the first-in entry and last-in entry in the RX FIFO**

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{RXFIFO base} + 4 \times (\text{POPXPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{RXFIFO base} + 4 \times [(\text{RXCTR} + \text{POPXPTR} - 1) \text{ modulo RXFIFO depth}]$$

where:

RXFIFO base = base address of receive FIFO

RXCTR = receive FIFO counter

POPNXPTR = pop next pointer

RX FIFO depth = receive FIFO depth, implementation specific



## 27 Timers

### 27.1 Introduction

This chapter describes the timer modules implemented on the microcontroller:

- System Timer Module (STM)
- Enhanced Modular IO Subsystem (eMIOS)
- Periodic Interrupt Timer (PIT)

The microcontroller also has a [Chapter 13: Real Time Clock / Autonomous Periodic Interrupt \(RTC/API\)](#) module. The main purpose of this is to provide a periodic device wakeup source.

### 27.2 Technical overview

This section gives a technical overview of each of the timers as well as detailing the pins that can be used to access the timer peripherals if applicable.

[Figure 351](#) details the interaction between the timers and the eDMA, INTC, CTU, and ADC.

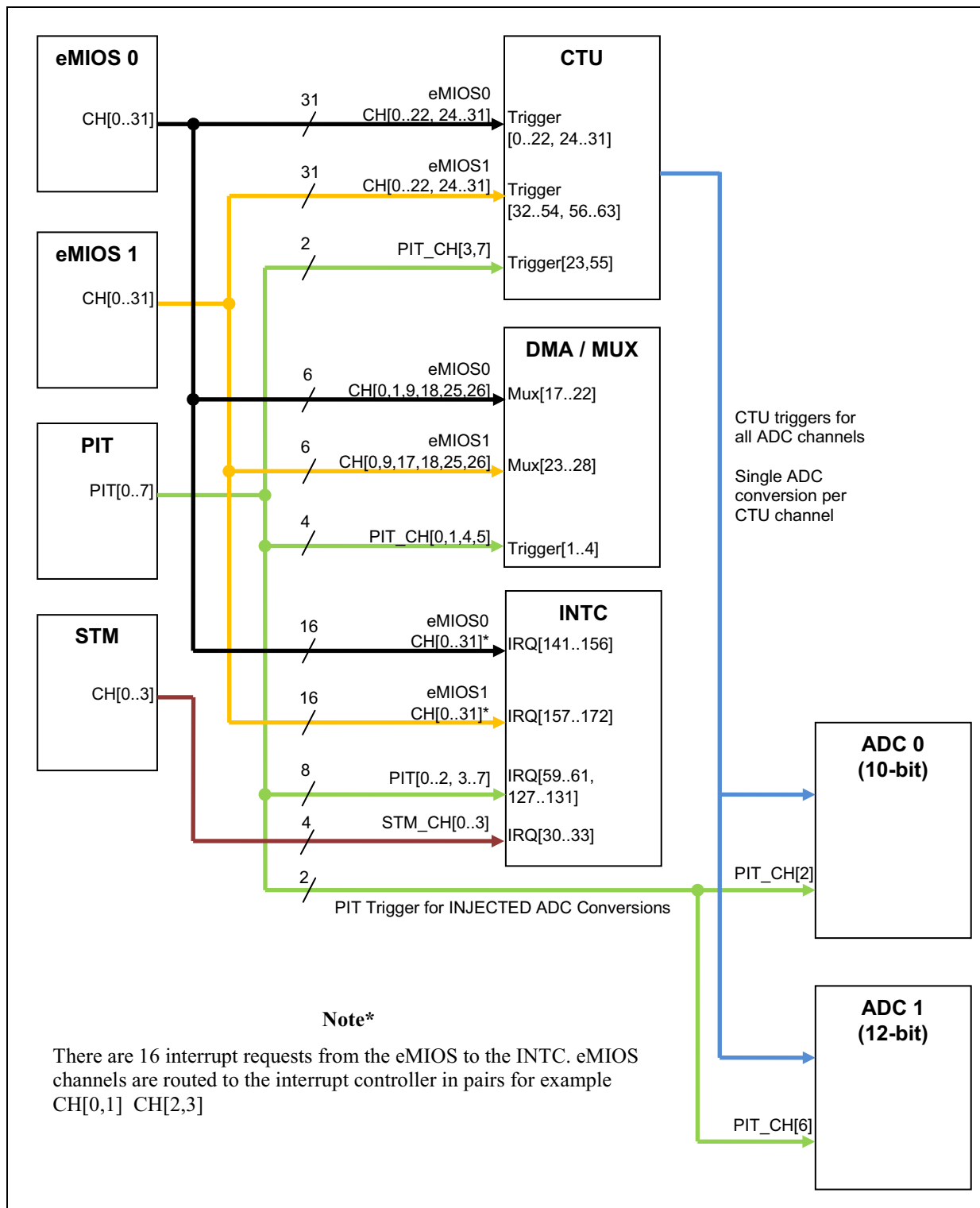


Figure 351. Interaction between timers and relevant peripherals

### 27.2.1 Overview of the STM

The STM is a 32-bit free running up-counter clocked by the system clock with a configurable 8-bit clock pre-scaler (divide by 1 to 256). The counter is disabled out of reset and must therefore be enabled by software prior to use. The counter value can be read at any time.

The STM has four 32-bit compare channels. Each channel can generate a unique interrupt on an exact match event with the free running counter.

The STM is often used to analyse code execution times. By starting the STM and reading the timer before and after a task or function, you can make an accurate measurement of the time taken in clock cycles to perform the task.

The STM can be configured to stop (freeze) or continue to run in debug mode and is available for use in all operating mode where the system clock is present (not STANDBY or certain STOP mode configurations)

There are no external pins associated with the STM.

### 27.2.2 Overview of the eMIOS

Each eMIOS offers a combination of PWM, Output Capture and Input Compare functions. There are different types of channel implemented and not every channel supports every eMIOS function. The channel functionality also differs between each eMIOS module. See [Section 27.4: Enhanced Modular IO Subsystem \(eMIOS\)](#), for more details.

Each channel has its own independent 16-bit counter. To allow synchronization between channels, there are a number of shared counter busses that can be used as a common timing reference. These counter buses can be used in combination with the individual channel counters to provide advanced features such as center aligned PWM with dead time insertion.

Once configured, the eMIOS needs very little CPU intervention. Interrupts, eDMA requests and CTU trigger requests can be raised based on eMIOS flag and timeout events.

The eMIOS is clocked from the system clock via peripheral clock group 3 (with a maximum permitted clock frequency of 64 MHz). The eMIOS can be used in all modes where the system clock is available (which excludes STANDBY mode and STOP mode when the system clock is turned off). The eMIOS has an option to allow the eMIOS counters to freeze or to continue running in debug mode.

The CTU allows an eMIOS event to trigger a single ADC conversion via the CTU without any CPU intervention. Without the CTU, the eMIOS would have to trigger an interrupt request. The respective ISR would then perform a software triggered ADC conversion. This not only uses CPU resource, but also increases the latency between the eMIOS event and the ADC trigger.

The eMIOS "Output Pulse Width Modulation with Trigger" mode (see [Section 27.4.4.4: Output Pulse Width Modulation with Trigger \(OPWMT\) mode](#)) allows a customisable trigger point to be defined at any point in the waveform period. This is extremely useful for LED lighting applications where the trigger can be set to a point where the PWM output is high but after the initial inrush current to the LED has occurred. The PWM trigger can then cause the CTU to perform a single ADC conversion which in turn measures the operating conditions of the LED to ensure it is working within specification. A watchdog feature on the ADC allows channels to be monitored and if the results fall outwith a specific range an interrupt is triggered. This means that all of the measurement is without CPU intervention if the results are within range.

To make it easier to plan which pins to use for the eMIOS, [Table 344](#) and [Table 345](#) show the eMIOS channel numbers that are available on each pin. The color shading matches the channel configuration diagram in the eMIOS section.

**Table 344. eMIOS\_0 channel to pin mapping**

Channel	Pin function			Channel	Pin function		
	ALT1	ALT2	ALT3		ALT1	ALT2	ALT3
UC[0]	PA[0]		PA[14]	UC[16]	PE[0]		
UC[1]	PA[1]		PA[15]	UC[17]	PE[1]		
UC[2]	PA[2]			UC[18]	PE[2]		
UC[3]	PA[3], PB[11]	PC[8]		UC[19]	PE[3]		
UC[4]	PA[4], PB[12]			UC[20]	PE[4]		
UC[5]	PA[5], PB[13]			UC[21]	PE[5]		
UC[6]	PA[6], PB[14]			UC[22]	PE[6], PF[5]	PE[8]	
UC[7]	PA[7], PB[15]	PC[9]		UC[23]	PE[7], PF[6]	PE[9]	
UC[8]	PA[8]			UC[24]	PE[11], PG[10]	PD[12]	
UC[9]	PA[9]			UC[25]	PG[11]	PD[13]	
UC[10]	PA[10], PF[0]			UC[26]	PG[12]	PD[14]	
UC[11]	PA[11], PF[1]			UC[27]	PG[13]	PD[15]	
UC[12]	PC[12], PF[2]			UC[28]	PI[0]	PA[12]	
UC[13]	PC[13], PF[3]		PA[0]	UC[29]	PI[1]	PA[13]	
UC[14]	PC[14], PF[4]	PA[8]		UC[30]	PI[2]	PB[0]	PB[2]
UC[15]	PC[15]			UC[31]	PB[3], PI[3]	PB[1]	

**Table 345. eMIOS\_1 channel to pin mapping**

Channel	Pin function			Channel	Pin function		
	ALT1	ALT2	ALT3		ALT1	ALT2	ALT3
UC[0]	PG[14]			UC[16]	PG[7]		
UC[1]	PF[9], PG[15]			UC[17]	PG[8]		PH[15]
UC[2]	PH[0]		PF[10]	UC[18]	PG[9]	PJ[4]	
UC[3]	PH[1]	PF[11]		UC[19]		PE[12]	
UC[4]	PF[15], PH[2]			UC[20]		PE[13]	
UC[5]	PH[3]		PH[11]	UC[21]		PE[14]	
UC[6]	PH[4]			UC[22]		PE[15]	
UC[7]	PH[5]			UC[23]		PG[0]	
UC[8]	PH[6]			UC[24]		PG[1]	
UC[9]	PH[7]			UC[25]	PF[12]		PH[12]
UC[10]	PH[8]			UC[26]	PF[13]		PH[13]
UC[11]	PG[2]			UC[27]		PF[14]	PH[14]
UC[12]	PG[3]			UC[28]	PI[4]	PC[6]	
UC[13]	PG[4]			UC[29]	PI[5]	PC[7]	
UC[14]	PG[5]			UC[30]	PI[6]	PG[7]	PE[10]
UC[15]	PG[6]			UC[31]	PC[4], PI[7]	PG[10]	

### 27.2.3 Overview of the PIT

The PIT module consists of 8 Periodic Interrupt Timers (PITs) clocked from the system clock.

Out of reset, the PITs are disabled. There is a global disable control bit for all of the PIT timers. Before using the timers, software must clear the appropriate disabled bit. Each of the PIT timers are effectively standalone entities and each have their own timer and control registers.

The PIT timers are 32-bit count down timers. To use them, you must first program an initial value into the LDVAL register. The timer will then start to count down and can be read at any time. Once the timer reaches 0x0000\_0000, a flag is set and the previous value is automatically re-loaded into the LDVAL register and the countdown starts again. The flag event can be routed to a dedicated INTC interrupt if desired.

The PIT is also used to trigger other events:

- 4 of the PIT channels can be used as an eDMA trigger
- 2 PIT channel can be used to trigger a CTU ADC conversion (single)
- 2 PIT channels (1 per ADC module) can be used to directly trigger injected conversions on the ADC

The timers can be configured to stop (freeze) or to continue to run in debug mode. The PITs are available in all modes where a system clock is generated.

There are no external pins associated with the PIT.

## 27.3 System Timer Module (STM)

### 27.3.1 Introduction

#### 27.3.1.1 Overview

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

#### 27.3.1.2 Features

The STM has the following features:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

#### 27.3.1.3 Modes of operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM\_CR register. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

### 27.3.2 External signal description

The STM does not have any external interface signals.

### 27.3.3 Memory map and register definition

The STM programming model has fourteen 32-bit registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generates a bus error termination.

#### 27.3.3.1 Memory map

The STM memory map is shown in [Table 346](#).

**Table 346. STM memory map**

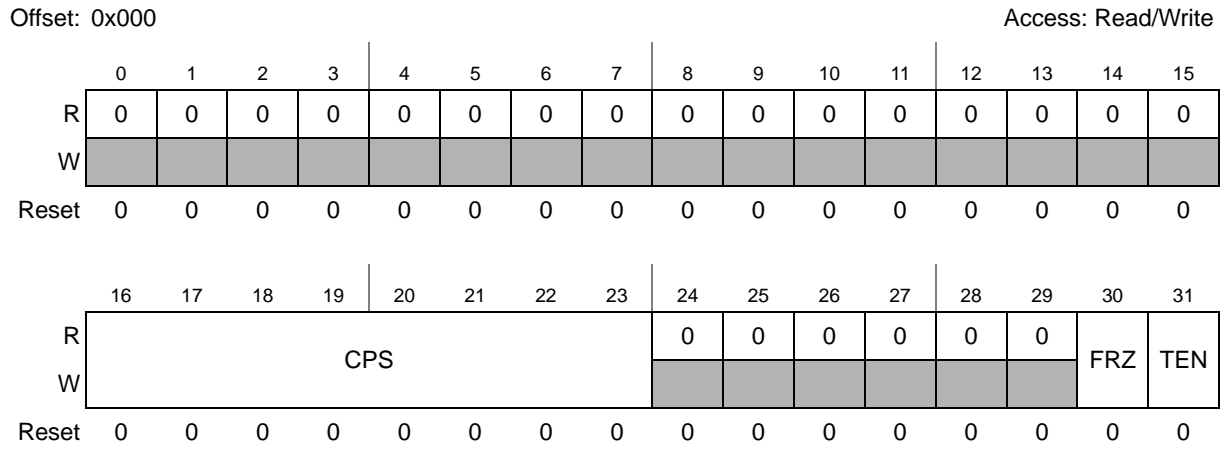
Base address: 0xFF3_C000		
Address offset	Register	Location
0x0000	STM Control Register (STM_CR)	<a href="#">on page 695</a>
0x0004	STM Counter Value (STM_CNT)	<a href="#">on page 695</a>
0x0008–0x000C	Reserved	
0x0010	STM Channel 0 Control Register (STM_CCR0)	<a href="#">on page 696</a>
0x0014	STM Channel 0 Interrupt Register (STM_CIR0)	<a href="#">on page 696</a>
0x0018	STM Channel 0 Compare Register (STM_CMP0)	<a href="#">on page 697</a>
0x001C	Reserved	
0x0020	STM Channel 1 Control Register (STM_CCR1)	<a href="#">on page 696</a>
0x0024	STM Channel 1 Interrupt Register (STM_CIR1)	<a href="#">on page 696</a>
0x0028	STM Channel 1 Compare Register (STM_CMP1)	<a href="#">on page 697</a>
0x002C	Reserved	
0x0030	STM Channel 2 Control Register (STM_CCR2)	<a href="#">on page 696</a>
0x0034	STM Channel 2 Interrupt Register (STM_CIR2)	<a href="#">on page 696</a>
0x0038	STM Channel 2 Compare Register (STM_CMP2)	<a href="#">on page 697</a>
0x003C	Reserved	
0x0040	STM Channel 3 Control Register (STM_CCR3)	<a href="#">on page 696</a>
0x0044	STM Channel 3 Interrupt Register (STM_CIR3)	<a href="#">on page 696</a>
0x0048	STM Channel 3 Compare Register (STM_CMP3)	<a href="#">on page 697</a>
0x004C–0x3FFF	Reserved	

#### 27.3.3.2 Register descriptions

The following sections detail the individual registers within the STM programming model.

**27.3.3.2.1 STM Control Register (STM\_CR)**

The STM Control Register (STM\_CR) includes the prescale value, freeze control and timer enable bits.



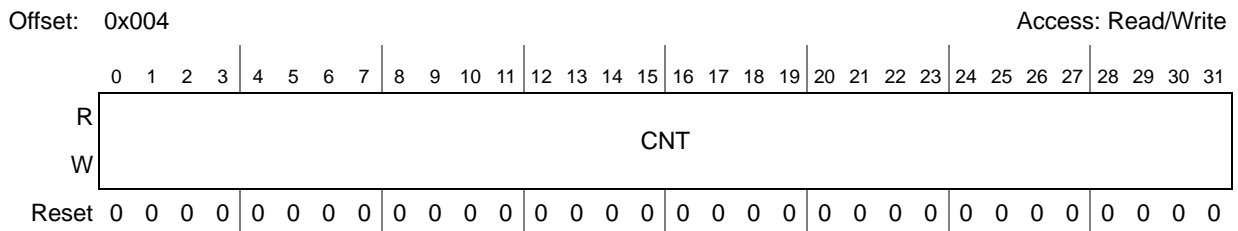
**Figure 352. STM Control Register (STM\_CR)**

**Table 347. STM\_CR field descriptions**

Field	Description
CPS	Counter Prescaler. Selects the clock divide value for the prescaler (1 - 256). 0x00 = Divide system clock by 1 0x01 = Divide system clock by 2 ... 0xFF = Divide system clock by 256
FRZ	Freeze. Allows the timer counter to be stopped when the device enters debug mode. 0 = STM counter continues to run in debug mode. 1 = STM counter is stopped in debug mode.
TEN	Timer Counter Enabled. 0 = Counter is disabled. 1 = Counter is enabled.

**27.3.3.2.2 STM Count Register (STM\_CNT)**

The STM Count Register (STM\_CNT) holds the timer count value.



**Figure 353. STM Count Register (STM\_CNT)**

**Table 348. STM\_CNT field descriptions**

Field	Description
CNT	Timer count value used as the time base for all channels. When enabled, the counter increments at the rate of the system clock divided by the prescale value.

**27.3.3.2.3 STM Channel Control Register (STM\_CCRn)**

The STM Channel Control Register (STM\_CCRn) has the enable bit for channel n of the timer.

Offset: 0x10+0x10\*n

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																CEN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 354. STM Channel Control Register (STM\_CCRn)**

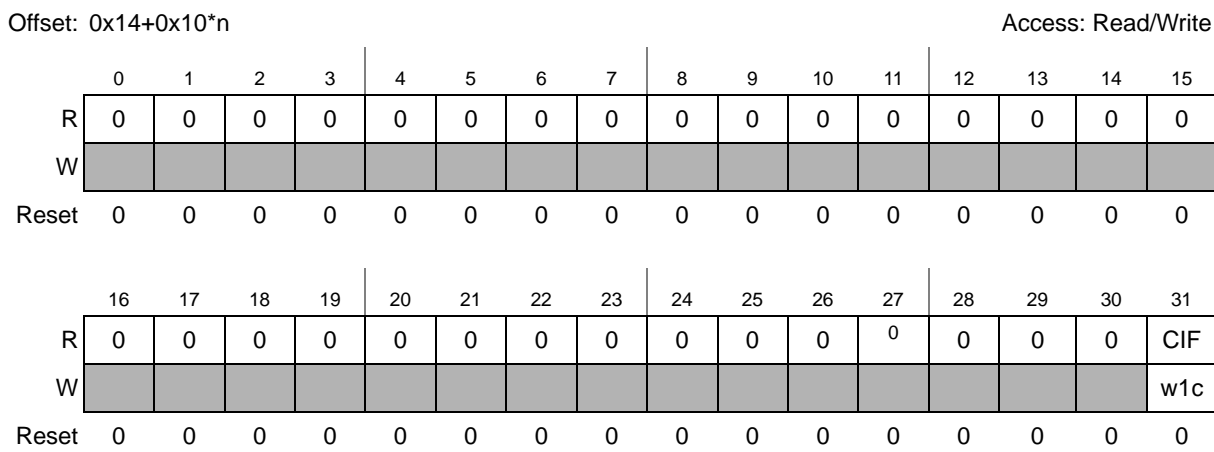
**Table 349. STM\_CCRn field descriptions**

Field	Description
CEN	Channel Enable. 0 = The channel is disabled. 1 = The channel is enabled.

**27.3.3.2.4 STM Channel Interrupt Register (STM\_CIRn)**

The STM Channel Interrupt Register (STM\_CIRn) has the interrupt flag for channel n of the timer.





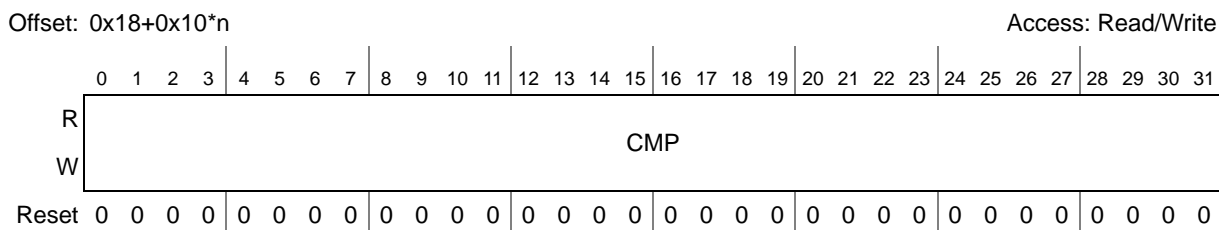
**Figure 355. STM Channel Interrupt Register (STM\_CIRn)**

**Table 350. STM\_CIRn field descriptions**

Field	Description
CIF	Channel Interrupt Flag 0 = No interrupt request. 1 = Interrupt request due to a match on the channel.

**27.3.3.2.5 STM Channel Compare Register (STM\_CMPn)**

The STM channel compare register (STM\_CMPn) holds the compare value for channel n.



**Figure 356. STM Channel Compare Register (STM\_CMPn)**

**Table 351. STM\_CMPn field descriptions**

Field	Description
CMP	Compare value for channel n. If the STM_CCRn[CEN] bit is set and the STM_CMPn register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIRn[CIF] bit is set.

**27.3.4 Functional description**

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM\_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM\_CR[CPS] field sets the divider to any value in the range from 1



to 256. The counter is enabled with the STM\_CR[TEN] bit. When enabled in normal mode the counter continuously increments. When enabled in debug mode the counter operation is controlled by the STM\_CR[FRZ] bit. When the STM\_CR[FRZ] bit is set, the counter is stopped in debug mode, otherwise it continues to run in debug mode. The counter rolls over at 0xFFFF\_FFFF to 0x0000\_0000 with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM\_CCRn), a channel interrupt register (STM\_CIRn) and a channel compare register (STM\_CMPn). The channel is enabled by setting the STM\_CCRn[CEN] bit. When enabled, the channel will set the STM\_CIRn[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a 1 to the STM\_CIRn[CIF] bit. A write of 0 to the STM\_CIRn[CIF] bit has no effect.

*Note:* STM counter does not advance when the system clock is stopped.

## 27.4 Enhanced Modular IO Subsystem (eMIOS)

### 27.4.1 Introduction

#### 27.4.1.1 Overview of the eMIOS module

The eMIOS provides functionality to generate or measure time events. Each channel provides a subset of the functionality available in the unified channel, at a resolution of 16 bits, and provides a user interface that is consistent with previous eMIOS implementations.

#### 27.4.1.2 Features of the eMIOS module

- 2 eMIOS blocks with 32 channels each
  - All 64 channels with OPWMT, which can be connected to the CTU
  - Both eMIOS blocks can be synchronized
- 1 global prescaler
- 16-bit data registers
- 10 x 16-bit wide counter buses
  - Counter buses B, C, D, and E can be driven by Unified Channel 0, 8, 16, and 24, respectively
  - Counter bus A is driven by the Unified Channel #23
  - Several channels have their own time base, alternative to the counter buses
  - Shared timebases through the counter buses
  - Synchronization among timebases
- Control and Status bits grouped in a single register
- Shadow FLAG register
- State of the UC can be frozen for debug purposes
- Motor control capability

### 27.4.1.3 Modes of operation

The Unified Channels can be configured to operate in the following modes:

- General purpose input/output
- Single Action Input Capture
- Single Action Output Compare
- Input Pulse Width Measurement
- Input Period Measurement
- Double Action Output Compare
- Modulus Counter
- Modulus Counter Buffered
- Output Pulse Width and Frequency Modulation Buffered
- Output Pulse Width Modulation Buffered
- Output Pulse Width Modulation with Trigger
- Center Aligned Output Pulse Width Modulation Buffered

These modes are described in [Section 27.4.4.1.1: UC modes of operation](#).

Each channel can have a specific set of modes implemented, according to device requirements.

If an unimplemented mode (reserved) is selected, the results are unpredictable such as writing a reserved value to MODE[0:6] in [Section 27.4.3.2.8: eMIOS UC Control Register \(EMIOSC\[n\]\)](#).

### 27.4.1.4 Channel implementation

[Figure 357](#) shows the channel configuration of the eMIOS blocks.

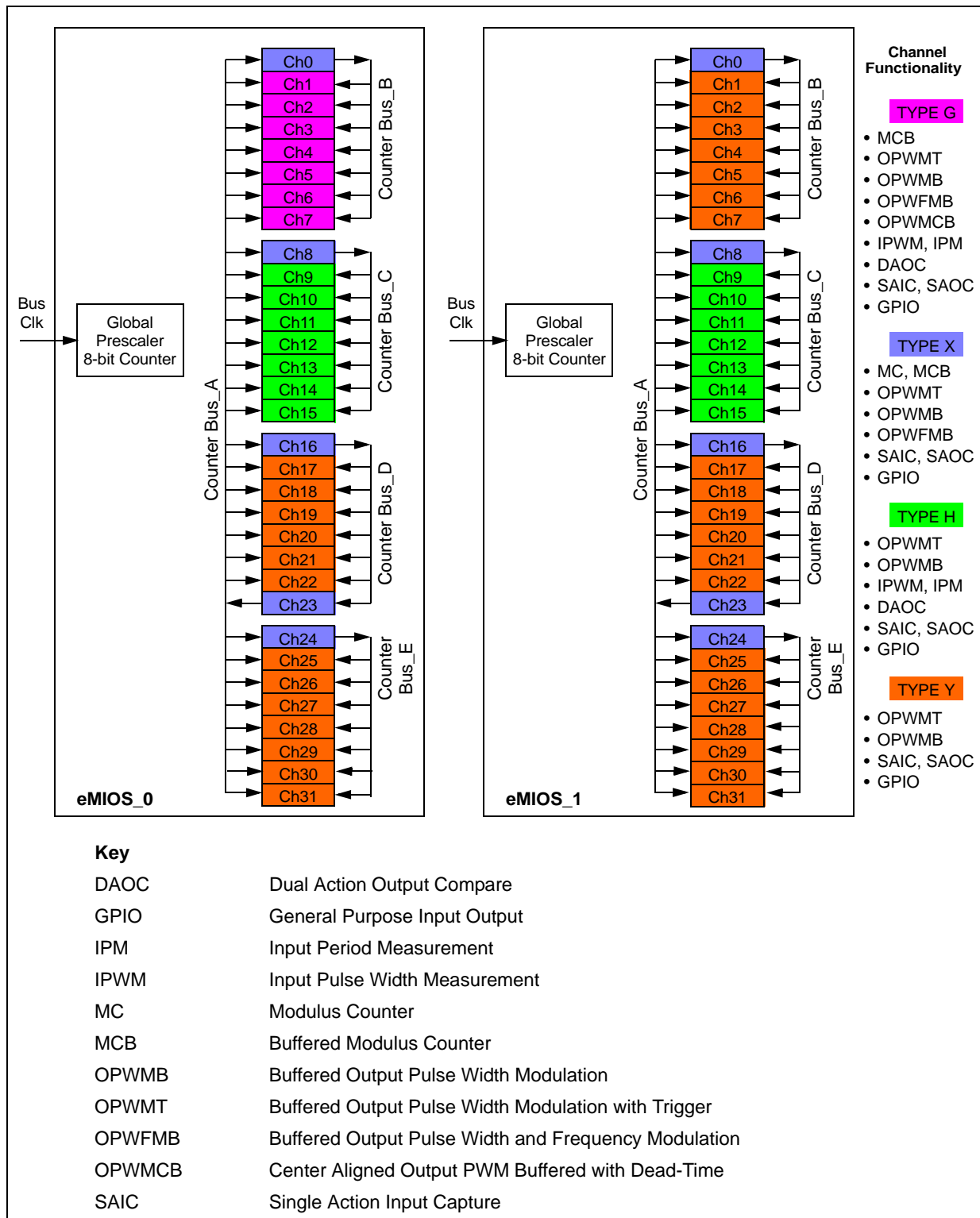


Figure 357. Channel configuration

**27.4.1.4.1 Channel mode selection**

Channel modes are selected using the mode selection bits MODE[0:6] in the [Section 27.4.3.2.8: eMIOS UC Control Register \(EMIOSC\[n\]\)](#). [Table 364](#) provides the specific mode selection settings for the eMIOS implementation on this device.

**27.4.2 External signal description**

For information on eMIOS external signals on this device, please refer to the signal description chapter of the reference manual.

**27.4.3 Memory map and register description**

**27.4.3.1 Memory maps**

The overall address map organization is shown in [Table 352](#).

**27.4.3.1.1 Unified Channel memory map**

**Table 352. eMIOS memory map**

Base addresses: 0xC3FA_0000 (eMIOS_0) 0xC3FA_4000 (eMIOS_1)		
Address offset	Description	Location
0x000–0x003	eMIOS Module Configuration Register (EMIOSMCR)	<a href="#">on page 702</a>
0x004–0x007	eMIOS Global FLAG (EMIOSGFLAG) Register	<a href="#">on page 703</a>
0x008–0x00B	eMIOS Output Update Disable (EMIOSOUDIS) Register	<a href="#">on page 704</a>
0x00C–0x00F	eMIOS Disable Channel (EMIOSUCDIS) Register	<a href="#">on page 705</a>
0x010–0x01F	Reserved	—
0x020–0x11F	Channel [0] to Channel [7]	—
0x120–0x21F	Channel [8] to Channel [15]	—
0x220–0x31F	Channel [16] to Channel [23]	—
0x320–0x41F	Channel [24] to Channel [31]	—
0x420–0xFFFF	Reserved	—

Addresses of Unified Channel registers are specified as offsets from the channel’s base address; otherwise the eMIOS base address is used as reference.

Table 353 describes the Unified Channel memory map.

**Table 353. Unified Channel memory map**

UC[n] base address	Description	Location
0x00	eMIOS UC A Register (EMIOSA[n])	on page 706
0x04	eMIOS UC B Register (EMIOSB[n])	on page 706
0x08	eMIOS UC Counter Register (EMIOSCNT[n])	on page 707
0x0C	eMIOS UC Control Register (EMIOSC[n])	on page 708
0x10	eMIOS UC Status Register (EMIOSS[n])	on page 712
0x14	eMIOS UC Alternate A Register (EMIOSALTA[n])	on page 713
0x18–0x1F	Reserved	—

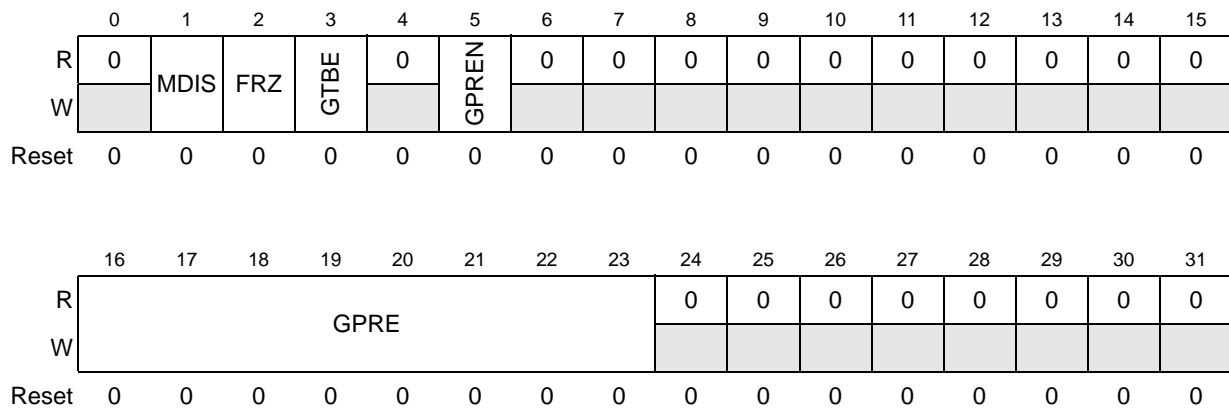
**27.4.3.2 Register description**

All control registers are 32 bits wide. Data registers and counter registers are 16 bits wide.

**27.4.3.2.1 eMIOS Module Configuration Register (EMIOSMCR)**

The EMIOSMCR contains global control bits for the eMIOS block.

Address: eMIOS base address +0x00



**Figure 358. eMIOS Module Configuration Register (EMIOSMCR)**

**Table 354. EMIOSMCR field descriptions**

Field	Description
MDIS	Module Disable Puts the eMIOS in low power mode. The MDIS bit is used to stop the clock of the block, except the access to registers EMIOSMCR, EMIOSOUDIS and EMIOSUCDIS. 1 = Enter low power mode 0 = Clock is running

**Table 354. EMIOSMCR field descriptions (continued)**

Field	Description
FRZ	<p>Freeze</p> <p>Enables the eMIOS to freeze the registers of the Unified Channels when Debug Mode is requested at MCU level. Each Unified Channel should have FREN bit set in order to enter freeze state. While in Freeze state, the eMIOS continues to operate to allow the MCU access to the Unified Channels registers. The Unified Channel will remain frozen until the FRZ bit is written to '0' or the MCU exits Debug mode or the Unified Channel FREN bit is cleared.</p> <p>1 = Stops Unified Channels operation when in Debug mode and the FREN bit is set in the EMIOSC[n] register                      0 = Exit freeze state</p>
GTBE	<p>Global Time Base Enable</p> <p>The GTBE bit is used to export a Global Time Base Enable from the module and provide a method to start time bases of several blocks simultaneously.</p> <p>1 = Global Time Base Enable Out signal asserted                      0 = Global Time Base Enable Out signal negated</p> <p><b>Note:</b> The Global Time Base Enable input pin controls the internal counters. When asserted, Internal counters are enabled. When negated, Internal counters disabled.</p>
GPREN	<p>Global Prescaler Enable</p> <p>The GPREN bit enables the prescaler counter.</p> <p>1 = Prescaler enabled                      0 = Prescaler disabled (no clock) and prescaler counter is cleared</p>
GPRE	<p>Global Prescaler</p> <p>The GPRE bits select the clock divider value for the global prescaler, as shown in <a href="#">Table 355</a>.</p>

**Table 355. Global prescaler clock divider**

GPRE	Divide ratio
00000000	1
00000001	2
00000010	3
00000011	4
.	.
.	.
.	.
.	.
11111110	255
11111111	256

**27.4.3.2.2 eMIOS Global FLAG (EMIOGFLAG) Register**

The EMIOGFLAG is a read-only register that groups the flag bits (F[31:0]) from all channels. This organization improves interrupt handling on simpler devices. Each bit relates to one channel.

For Unified Channels these bits are mirrors of the FLAG bits in the EMIOSS[n] register.



Address: eMIOS base address +0x04

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F31	F30	F29	F28	F27	F26	F25	F24	F23	F22	F21	F20	F19	F18	F17	F16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 359. eMIOS Global FLAG (EMIOSGFLAG) Register

Table 356. EMIOSGFLAG field descriptions

Field	Description
Fn	Channel [n] Flag bit

27.4.3.2.3 eMIOS Output Update Disable (EMIOSOUDIS) Register

Address: eMIOS base address +0x08

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OU31	OU30	OU29	OU28	OU27	OU26	OU25	OU24	OU23	OU22	OU21	OU20	OU19	OU18	OU17	OU16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OU15	OU14	OU13	OU12	OU11	OU10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 360. eMIOS Output Update Disable (EMIOSOUDIS) Register

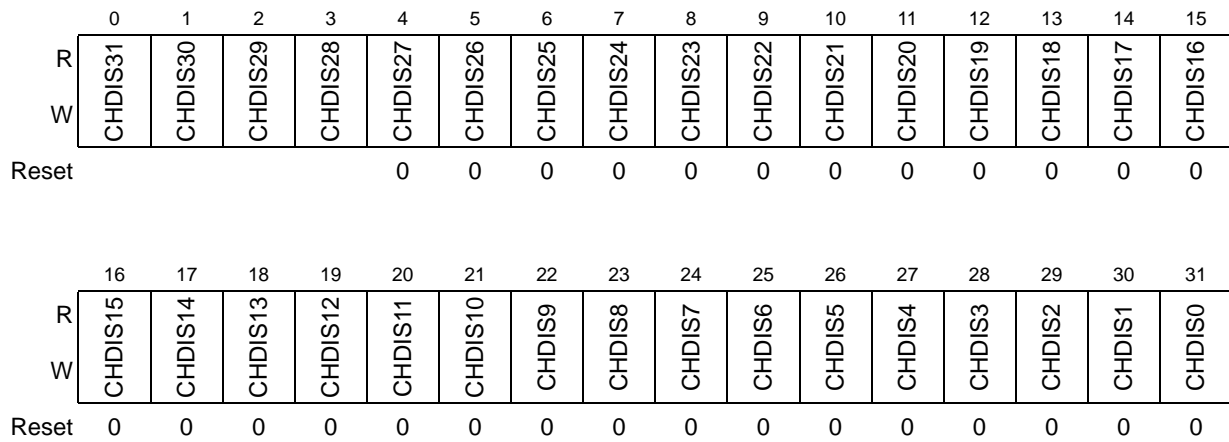


**Table 357. EMIOSOUDIS field descriptions**

Field	Description
OUn	Channel [n] Output Update Disable bit When running MC, MCB or an output mode, values are written to registers A2 and B2. OU[n] bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 1 = Transfers disabled 0 = Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately.

**27.4.3.2.4 eMIOS Disable Channel (EMIOSUCDIS) Register**

Address: eMIOS base address +0x0C



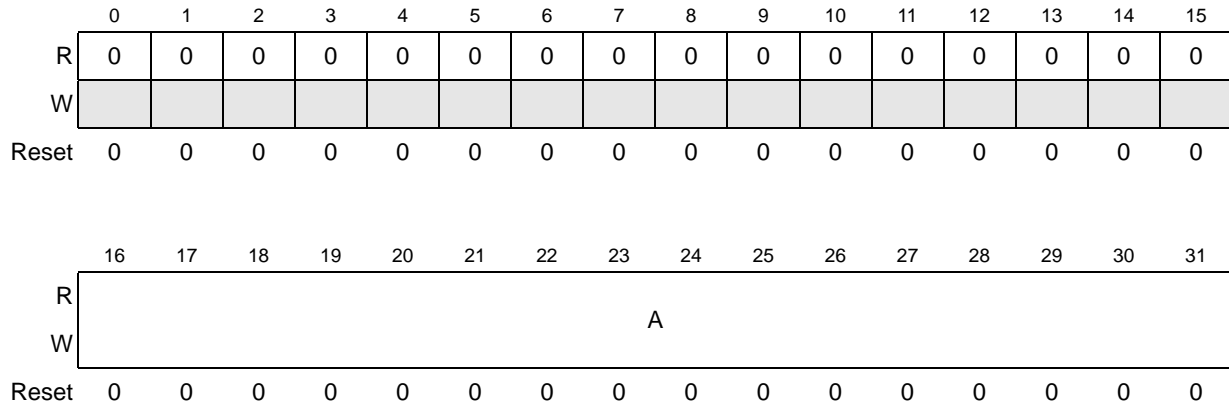
**Figure 361. eMIOS Enable Channel (EMIOSUCDIS) Register**

**Table 358. EMIOSUCDIS field descriptions**

Field	Description
CHDISn	Enable Channel [n] bit The CHDIS[n] bit is used to disable each of the channels by stopping its respective clock. 1 = Channel [n] disabled 0 = Channel [n] enabled

### 27.4.3.2.5 eMIOS UC A Register (EMIOSA[n])

Address: UC[n] base address + 0x00

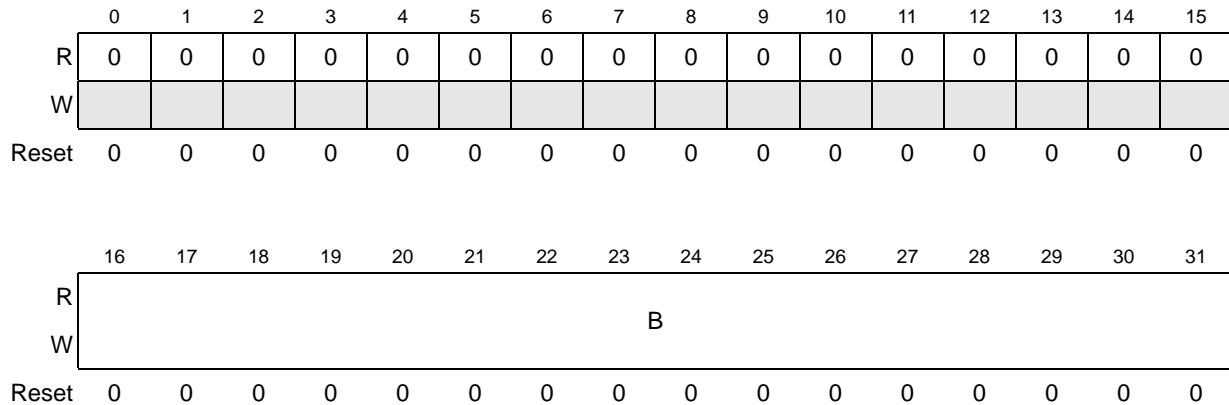


**Figure 362. eMIOS UC A Register (EMIOSA[n])**

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOSA[n]. Both A1 and A2 are cleared by reset. [Figure 359](#) summarizes the EMIOSA[n] writing and reading accesses for all operation modes. For more information see [Section 27.4.4.1.1: UC modes of operation](#).

### 27.4.3.2.6 eMIOS UC B Register (EMIOSB[n])

Address: UC[n] base address + 0x04



**Figure 363. eMIOS UC B Register (EMIOSB[n])**

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOSB[n]. Both B1 and B2 are cleared by reset. [Table 359](#) summarizes the EMIOSB[n] writing and reading accesses for all operation modes. For more information see [Section 27.4.4.1.1: UC modes of operation](#).

Depending on the channel configuration, it may have EMIOSB register or not. This means that, if at least one mode that requires the register is implemented, then the register is present; otherwise it is absent.

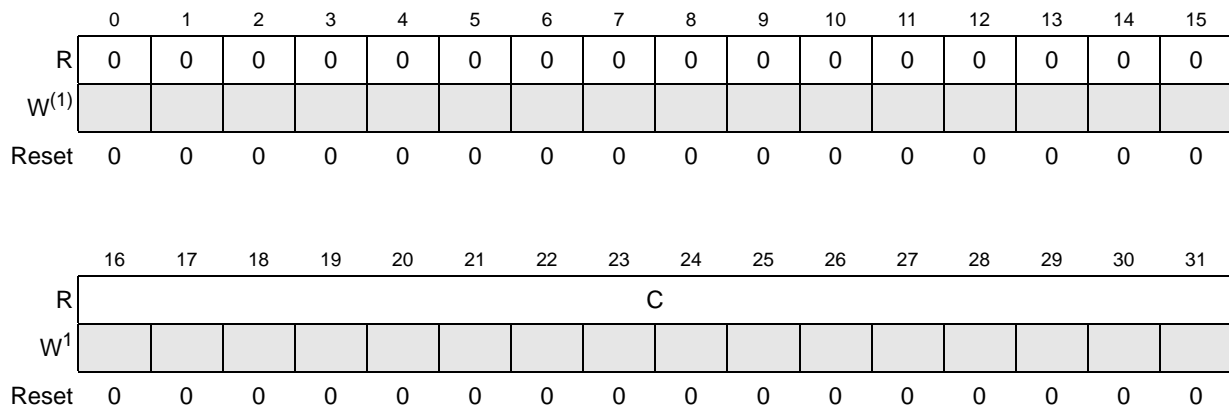
**Table 359. EMIOSA[n], EMIOSB[n] and EMIOSALTA[n] values assignment**

Operation mode	Register access					
	write	read	write	read	alt write	alt read
GPIO	A1, A2	A1	B1,B2	B1	A2	A2
SAIC <sup>1</sup>	—	A2	B2	B2	—	—
SAOC <sup>(1)</sup>	A2	A1	B2	B2	—	—
IPWM	—	A2	—	B1	—	—
IPM	—	A2	—	B1	—	—
DAOC	A2	A1	B2	B1	—	—
MC <sup>1</sup>	A2	A1	B2	B2	—	—
OPWMT	A1	A1	B2	B1	A2	A2
MCB <sup>1</sup>	A2	A1	B2	B2	—	—
OPWFMB	A2	A1	B2	B1	—	—
OPWMCB	A2	A1	B2	B1	—	—
OPWMB	A2	A1	B2	B1	—	—

1. In these modes, the register EMIOSB[n] is not used, but B2 can be accessed.

**27.4.3.2.7 eMIOS UC Counter Register (EMIOSCNT[n])**

Address: UC[n] base address + 0x08



**Figure 364. eMIOS UC Counter Register (EMIOSCNT[n])**

1. In GPIO mode or Freeze action, this register is writable.

The EMIOSCNT[n] register contains the value of the internal counter. When GPIO mode is selected or the channel is frozen, the EMIOSCNT[n] register is read/write. For all others modes, the EMIOSCNT[n] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to [Section 27.4.4.1.1: UC modes of operation](#) for details).

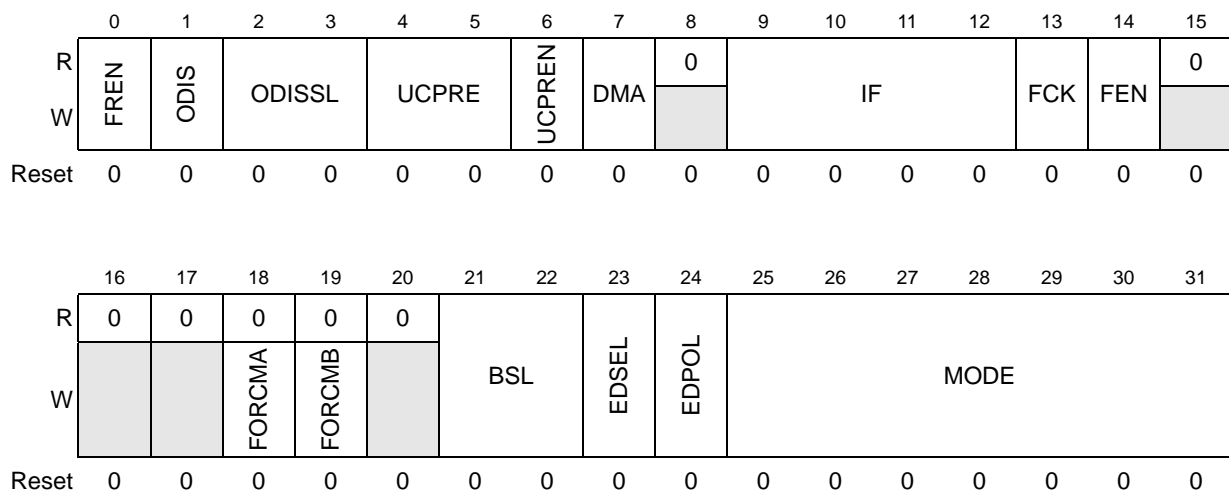
Depending on the channel configuration it may have an internal counter or not. It means that if at least one mode that requires the counter is implemented, then the counter is present; otherwise it is absent.

Channels of type X and G have the internal counter enabled, so their timebase can be selected by channel's BSL[1:0]=11:eMIOS\_A - channels 0 to 8, 16, 23 and 24, eMIOS\_B = channels 0, 8, 16, 23 and 24. Other channels from the above list don't have internal counters.

**27.4.3.2.8 eMIOS UC Control Register (EMIOSC[n])**

The Control register gathers bits reflecting the status of the UC input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

Address: UC[n] base address + 0x0C



**Figure 365. eMIOS UC Control Register (EMIOSC[n])**

**Table 360. EMIOSC[n] field descriptions**

Field	Description
FREN	<p>Freeze Enable bit</p> <p>The FREN bit, if set and validated by FRZ bit in EMIOSMCR register allows the channel to enter freeze state, freezing all registers values when in debug mode and allowing the MCU to perform debug functions.</p> <p>1 = Freeze UC registers values 0 = Normal operation</p>
ODIS	<p>Output Disable</p> <p>This bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode.</p> <p>1 = If the selected Output Disable Input signal is asserted, the output pin goes to EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other output modes, but the Unified Channel continues to operate normally (that is, it continues to produce FLAG and matches). When the selected Output Disable Input signal is negated, the output pin operates normally.</p> <p>0 = The output pin operates normally.</p>

**Table 360. EMIOSC[n] field descriptions (continued)**

Field	Description
ODISSL	<p>Output Disable select</p> <p>This field selects one of the four output disable input signals, as follows:</p> <p>00 = Output Disable Input 0</p> <p>01 = Output Disable Input 1</p> <p>10 = Output Disable Input 2</p> <p>11 = Output Disable Input 3</p>
UCPRE	<p>Prescaler bits</p> <p>The UCPRE bits select the clock divider value for the internal prescaler of Unified Channel, as shown in <a href="#">Table 361</a>.</p>
UCPREN	<p>Prescaler Enable bit</p> <p>The UCPREN bit enables the prescaler counter.</p> <p>1 = Prescaler enabled</p> <p>0 = Prescaler disabled (no clock)</p>
DMA	<p>Direct Memory Access bit</p> <p>The DMA bit selects if the FLAG generation will be used as an interrupt request, as a DMA request or as a CTU trigger. The choice between a DMA request or a CTU trigger is determined by the value of bit TM in the register CTU_EVTTCFGRx (refer to the CTU chapter of the reference manual).</p> <p>1 = Flag/overrun assigned to DMA request or CTU trigger</p> <p>0 = Flag/overrun assigned to interrupt request</p>
IF	<p>Input Filter</p> <p>The IF field controls the programmable input filter, selecting the minimum input pulse width that can pass through the filter, as shown in <a href="#">Table 362</a>. For output modes, these bits have no meaning.</p>
FCK	<p>Filter Clock select bit</p> <p>The FCK bit selects the clock source for the programmable input filter.</p> <p>1 = Main clock</p> <p>0 = Prescaled clock</p>
FEN	<p>FLAG Enable bit</p> <p>The FEN bit allows the Unified Channel FLAG bit to generate an interrupt signal or a DMA request signal or a CTU trigger signal (The type of signal to be generated is defined by the DMA bit).</p> <p>1 = Enable (FLAG will generate an interrupt request or DMA request or a CTU trigger)</p> <p>0 = Disable (FLAG does not generate an interrupt request or DMA request or a CTU trigger)</p>
FORCMA	<p>Force Match A bit</p> <p>For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect.</p> <p>1 = Force a match at comparator A</p> <p>0 = Has no effect</p> <p><b>Note:</b> For input modes, the FORCMA bit is not used and writing to it has no effect.</p>

Table 360. EMIOSC[n] field descriptions (continued)

Field	Description
FORCMB	<p>Force Match B bit</p> <p>For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect.</p> <p>1 = Force a match at comparator B 0 = Has not effect</p> <p><b>Note:</b> For input modes, the FORCMB bit is not used and writing to it has no effect.</p>
BSL	<p>Bus Select</p> <p>The BSL field is used to select either one of the counter buses or the internal counter to be used by the Unified Channel. Refer to <a href="#">Table 363</a> for details.</p>
EDSEL	<p>Edge Selection bit</p> <p>For input modes, the EDSEL bit selects whether the internal counter is triggered by both edges of a pulse or just by a single edge as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.</p> <p>1 = Both edges triggering 0 = Single edge triggering defined by the EDPOL bit</p> <p>For GPIO in mode, the EDSEL bit selects if a FLAG can be generated.</p> <p>1 = No FLAG is generated 0 = A FLAG is generated as defined by the EDPOL bit</p> <p>For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.</p> <p>1 = The output flip-flop is toggled 0 = The EDPOL value is transferred to the output flip-flop</p>
EDPOL	<p>Edge Polarity bit</p> <p>For input modes, the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect.</p> <p>1 = Trigger on a rising edge 0 = Trigger on a falling edge</p> <p>For output modes, the EDPOL bit is used to select the logic level on the output pin.</p> <p>1 = A match on comparator A sets the output flip-flop, while a match on comparator B clears it 0 = A match on comparator A clears the output flip-flop, while a match on comparator B sets it</p>
MODE	<p>Mode selection</p> <p>The MODE field selects the mode of operation of the Unified Channel, as shown in <a href="#">Table 364</a>.</p> <p><b>Note:</b> If a reserved value is written to mode the results are unpredictable.</p>

Table 361. UC internalprescaler clock divider

UCPRE	Divide ratio
00	1
01	2
10	3
11	4

**Table 362. UC input filter bits**

IF <sup>(1)</sup>	Minimum input pulse width [FLT_CLK periods]
0000	Bypassed <sup>(2)</sup>
0001	02
0010	04
0100	08
1000	16
all others	Reserved

1. Filter latency is 3 clock edges.
2. The input signal is synchronized before arriving to the digital filter.

**Table 363. UC BSL bits**

BSL	Selected bus
00	All channels: counter bus[A]
01	Channels 0 to 7: counter bus[B] Channels 8 to 15: counter bus[C] Channels 16 to 23: counter bus[D] Channels 24 to 27: counter bus[E]
10	Reserved
11	All channels: internal counter

**Table 364. Channel mode selection**

MODE <sup>(1)</sup>	Mode of operation
0000000	General purpose Input/Output mode (input)
0000001	General purpose Input/Output mode (output)
0000010	Single Action Input Capture
0000011	Single Action Output Compare
0000100	Input Pulse Width Measurement
0000101	Input Period Measurement
0000110	Double Action Output Compare (with FLAG set on B match)
0000111	Double Action Output Compare (with FLAG set on both match)
0001000 – 0001111	Reserved
001000b	Modulus Counter (Up counter with clear on match start)
001001b	Modulus Counter (Up counter with clear on match end)
00101bb	Modulus Counter (Up/Down counter)
0011000 – 0100101	Reserved
0100110	Output Pulse Width Modulation with Trigger
0100111 – 1001111	Reserved

**Table 364. Channel mode selection (continued)**

MODE <sup>(1)</sup>	Mode of operation
101000b	Modulus Counter Buffered (Up counter)
101001b	Reserved
10101bb	Modulus Counter Buffered (Up/Down counter)
10110b0	Output Pulse Width and Frequency Modulation Buffered
10110b1	Reserved
10111b0	Center Aligned Output Pulse Width Modulation Buffered (with trail edge dead-time)
10111b1	Center Aligned Output Pulse Width Modulation Buffered (with lead edge dead-time)
11000b0	Output Pulse Width Modulation Buffered
1100001 – 1111111	Reserved

1. b = adjust parameters for the mode of operation. Refer to [Section 27.4.4.1.1: UC modes of operation](#) for details.

**27.4.3.2.9 eMIOS UC Status Register (EMIOSS[n])**

Address: UC[n] base address + 0x10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	w1c															w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 366. eMIOS UC Status Register (EMIOSS[n])**

**Table 365. EMIOSS[n] field descriptions**

Field	Description
OVR	Overrun bit The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. 1 = Overrun has occurred 0 = Overrun has not occurred
OVFL	Overflow bit The OVFL bit indicates that an overflow has occurred in the internal counter. OVFL must be cleared by software writing a 1 to the OVFLC bit. 1 = An overflow had occurred 0 = No overflow



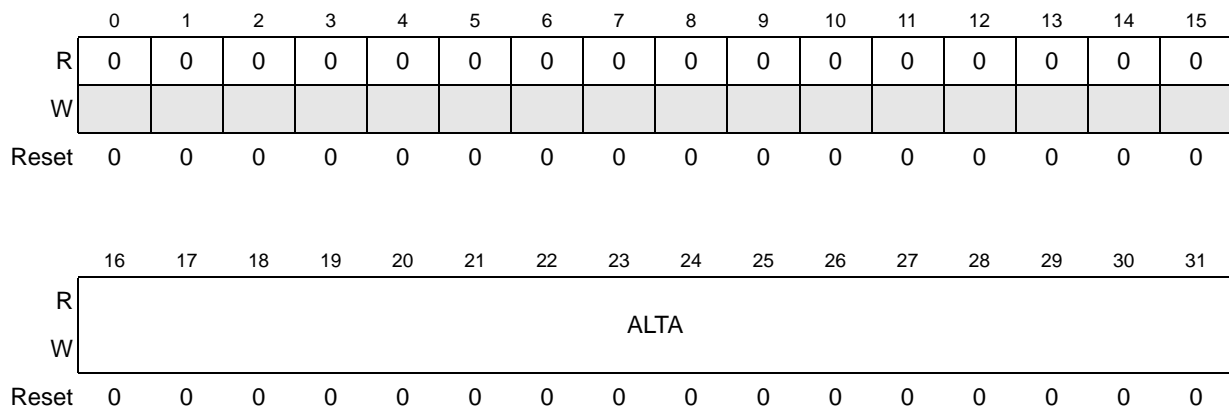


**Table 365. EMIOSS[n] field descriptions (continued)**

Field	Description
UCIN	Unified Channel Input pin bit The UCIN bit reflects the input pin state after being filtered and synchronized.
UCOUT	UCOUT — Unified Channel Output pin bit The UCOUT bit reflects the output pin state.
FLAG	FLAG bit The FLAG bit is set when an input capture or a match event in the comparators occurred. 1 = FLAG set event has occurred 0 = FLAG cleared <b>Note:</b> When DMA bit is set, the FLAG bit can be cleared by the DMA controller or the CTU.

**27.4.3.2.10 eMIOS UC Alternate A Register (EMIOSALTA[n])**

Address: UC[n] base address + 0x14



**Figure 367. eMIOS UC Alternate A register (EMIOSALTA[n])**

The EMIOSALTA[n] register provides an alternate address to access A2 channel registers in restricted modes (GPIO, OPWMT) only. If EMIOSA[n] register is used along with EMIOSALTA[n], both A1 and A2 registers can be accessed in these modes. [Figure 359](#) summarizes the EMIOSALTA[n] writing and reading accesses for all operation modes. Please, see Section , General purpose Input/Output (GPIO) mode, [Section 27.4.4.4: Output Pulse Width Modulation with Trigger \(OPWMT\) mode](#) for a more detailed description of the use of EMIOSALTA[n] register.

**27.4.4 Functional description**

The four types of channels of the eMIOS (types X, Y, G and H) can operate in the modes as listed in [Figure 357](#). The eMIOS provides independently operating unified channels (UC) that can be configured and accessed by a host MCU.

Up to three time bases<sup>(v)</sup> can be shared by the channels through five counter buses<sup>(w)</sup> and each unified channel can generate its own time base<sup>(x)</sup>. The eMIOS block is reset at positive edge of the clock (synchronous reset). All registers are cleared on reset.

### 27.4.4.1 Unified Channel (UC)

Each Unified Channel consists of:

- Counter bus selector, which selects the time base to be used by the channel for all timing functions
- A programmable clock prescaler
- Two double buffered data registers A and B that allow up to two input capture and/or output compare events to occur before software intervention is needed.
- Two comparators (equal only) A and B, which compares the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS Status and Control register

#### 27.4.4.1.1 UC modes of operation

The mode of operation of the Unified Channel is determined by the mode select bits MODE[0:6] in the eMIOS UC Control Register (EMIOSC[n]) (see [Figure 365](#) for details).

As the internal counter EMIOSCNT[n] continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

In order to provide smooth waveform generation even if A and B registers are changed on the fly, it is available the MCB, OPWFMB, OPWMB and OPWMCB modes. In these modes A and B registers are double buffered.

##### 27.4.4.1.1.1 General purpose Input/Output (GPIO) mode

In GPIO mode, all input capture and output compare functions of the UC are disabled, the internal counter (EMIOSCNT[n] register) is cleared and disabled. All control bits remain accessible. In order to prepare the UC for a new operation mode, writing to registers EMIOSA[n] or EMIOSB[n] stores the same value in registers A1/A2 or B1/B2, respectively. Writing to register EMIOSALTA[n] stores a value only in register A2.

MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

It is required that when changing MODE[0:6], the application software goes to GPIO mode first in order to reset the UC's internal functions properly. Failure to do this could lead to

- 
- v. Time bases can be supplied by:
    - channel 23 to all unified channels
    - channel 0 to channels 0 to 7, by channel 8 to channels 8 to 15, by channel 16 to channels 16 to 23, by channel 24 to channels 24 to 31
    - channel's internal counter when available.
  - w. Internal eMIOS architecture have one global counter bus A and four local counter buses B, C, D, and E, that distribute the time bases described in Note 1 (a) and (b).
  - x. Channels of type X and G have the internal counter enabled, so their timebase can be selected by channel's BSL[1:0]=11: eMIOS\_A - channels 0 to 8, 16, 23 and 24 eMIOS\_B = channels 0, 8, 16, 23 and 24.

invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

In GPIO input mode (MODE[0:6] = 0000000), the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

In GPIO output mode (MODE[0:6] = 0000001), the Unified Channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

### 27.4.4.1.1.2 Single Action Input Capture (SAIC) mode

In SAIC mode (MODE[0:6] = 0000010), when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. The FLAG bit is set along with the capture event to indicate that an input capture has occurred. Register EMIOSA[n] returns the value of register A2. As soon as the SAIC mode is entered coming out from GPIO mode the channel is ready to capture events. The events are captured as soon as they occur thus reading register A always returns the value of the latest captured event. Subsequent captures are enabled with no need of further reads from EMIOSA[n] register. The FLAG is set at any time a new event is captured.

The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOSC[n] register.

Figure 368 and Figure 369 show how the Unified Channel can be used for input capture.

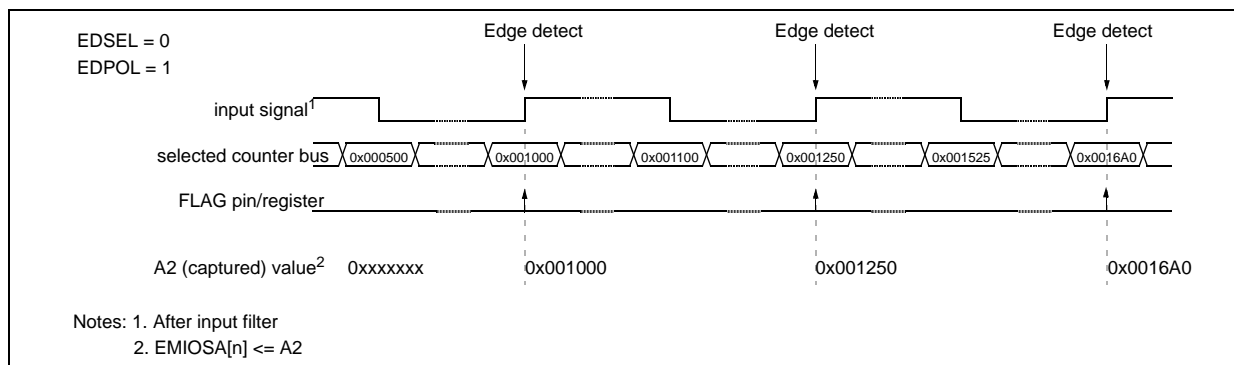
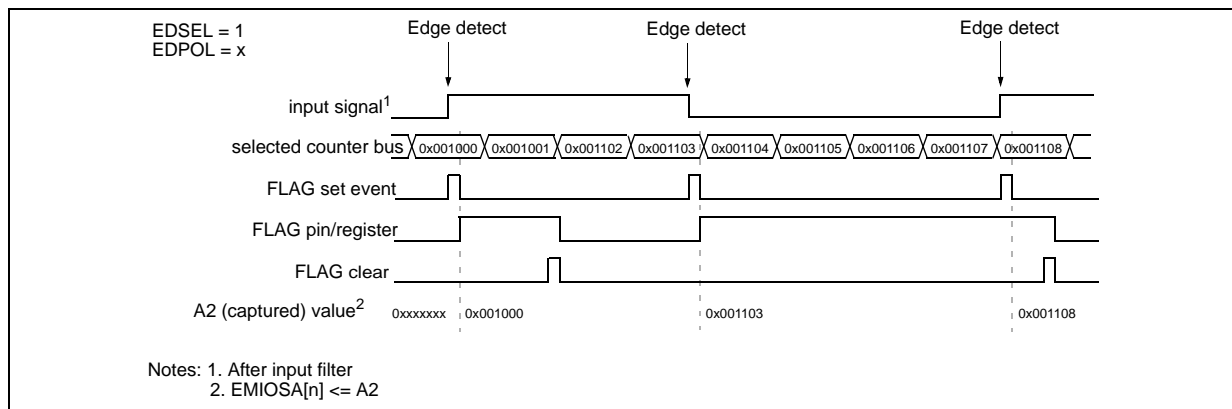


Figure 368. Single action input capture with rising edge triggering example



**Figure 369. Single action input capture with both edges triggering example**

### 27.4.4.1.1.3 Single Action Output Compare (SAOC) mode

In SAOC mode (MODE[0:6] = 0000011) a match value is loaded in register A2 and then immediately transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects whether the output flip-flop is toggled or the value in EDPOL is transferred to it. Along with the match the FLAG bit is set to indicate that the output compare match has occurred. Writing to register EMIOSA[n] stores the value in register A2 and reading to register EMIOSA[n] returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in EMIOSC[n] register. In this case, the FLAG bit is not set.

When SAOC mode is entered coming out from GPIO mode the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

Counter bus can be either internal or external and is selected through bits BSL[0:1].

[Figure 370](#) and [Figure 371](#) show how the Unified Channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively. Note that once in SAOC mode the matches are enabled thus the desired match value on register A1 must be written before the mode is entered. A1 register can be updated at any time thus modifying the match value which will reflect in the output signal generated by the channel. Subsequent matches are enabled with no need of further writes to EMIOSA[n] register. The FLAG is set at the same time a match occurs (see [Figure 372](#)).

*Note:* The channel internal counter in SAOC mode is free-running. It starts counting as soon as the SAOC mode is entered.

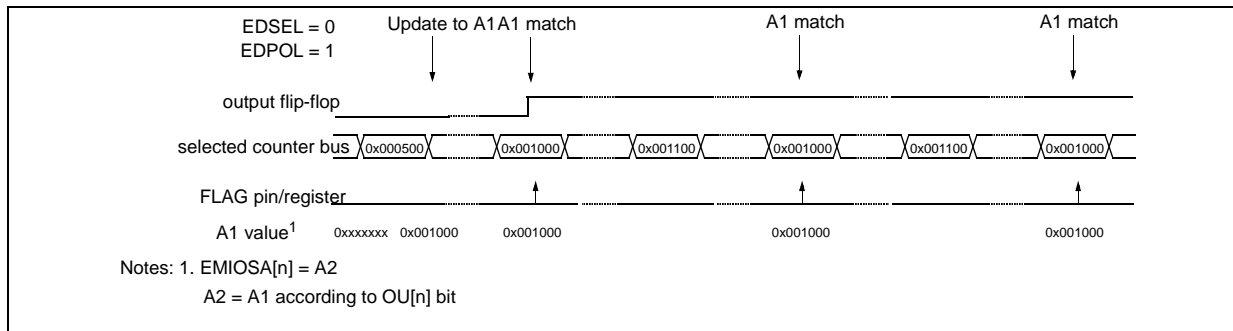


Figure 370. SAOC example with EDPOL value being transferred to the output flip-flop

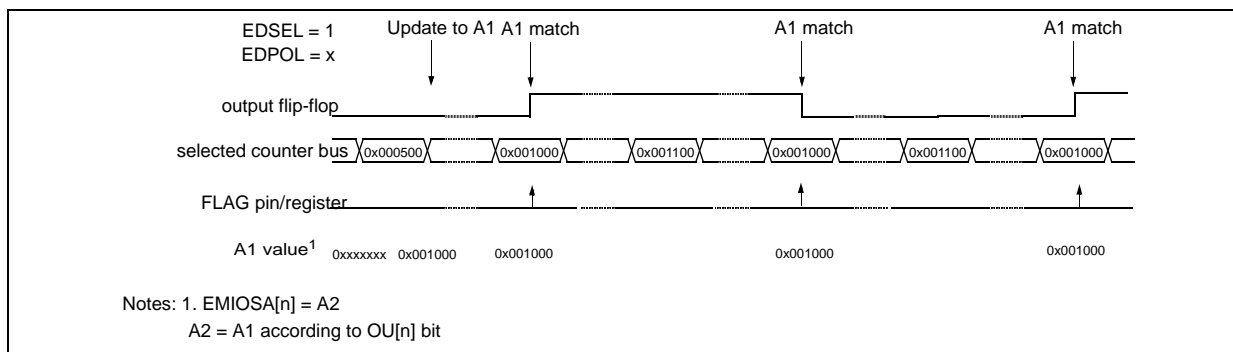


Figure 371. SAOC example toggling the output flip-flop

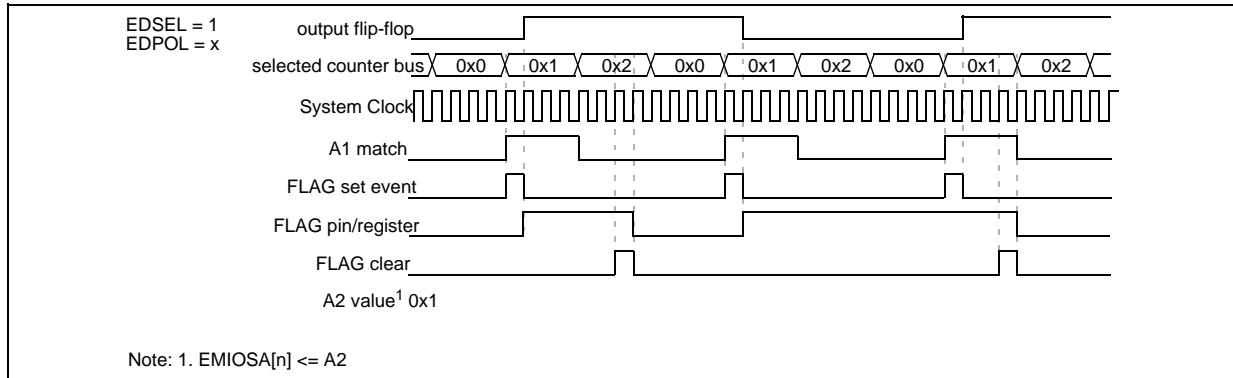


Figure 372. SAOC example with flag behavior

27.4.4.1.1.4 Input Pulse Width Measurement (IPWM) Mode

The IPWM mode (MODE[0:6] = 0000100) allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (that is, pulse polarity) is selected by EDPOL bit in the EMIOSC[n] register. Registers EMIOSA[n] and EMIOSB[n] return the values in register A2 and B1, respectively.

The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the

selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1 and A1 will be updated with the latest captured values and the FLAG will remain set. Registers EMIOSA[n] and EMIOSB[n] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOSA[n] forces B1 be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of EMIOSB[n] register. Reading EMIOSB[n] register forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 373 shows how the Unified Channel can be used for input pulse width measurement.

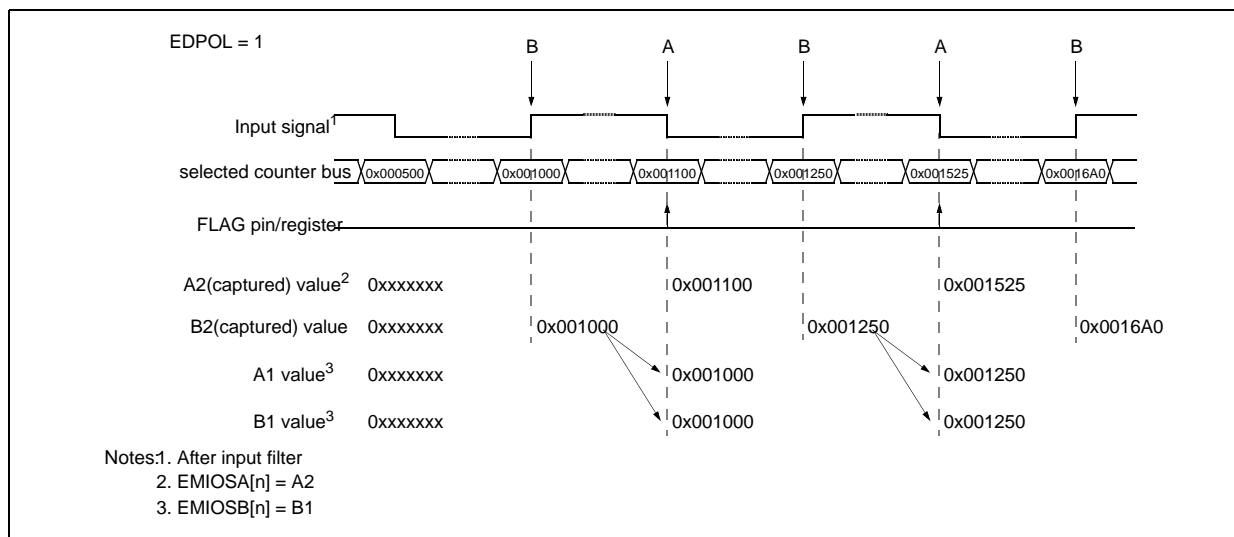
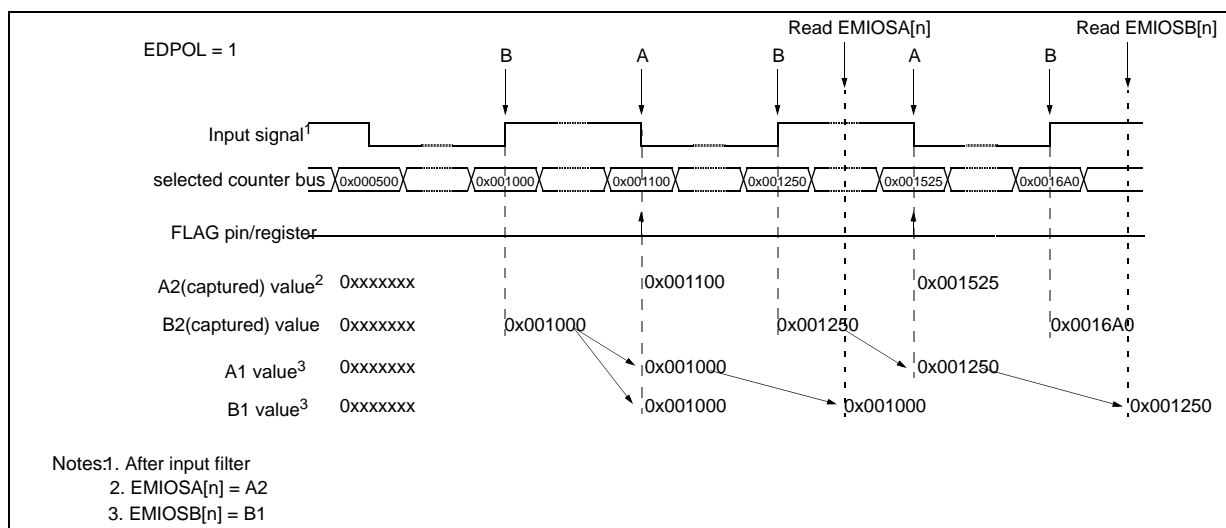


Figure 373. Input pulse width measurement example

Figure 374 shows the A1 and B1 updates when EMIOSA[n] and EMIOSB[n] register reads occur. Note that A1 register has always coherent data related to A2 register. Note also that when EMIOSA[n] read is performed B1 register is loaded with A1 register content. This guarantee that the data in register B1 has always the coherent data related to the last EMIOSA[n] read. The B1 register updates remains locked until EMIOSB[n] read occurs. If EMIOSA[n] read is performed B1 is updated with A1 register content even if B1 update is locked by a previous EMIOSA[n] read operation.



**Figure 374. B1 and A1 updates at EMIOSA[n] and EMIOSB[n] reads**

Reading EMIOSA[n] followed by EMIOSB[n] always provides coherent data. If not coherent data is required for any reason, the sequence of reads should be inverted, therefore EMIOSB[n] should be read prior to EMIOSA[n] register. Note that even in this case B1 register updates will be blocked after EMIOSA[n] read, thus a second EMIOSB[n] is required in order to release B1 register updates.

**27.4.4.1.1.5 Input Period Measurement (IPM) mode**

The IPM mode (MODE[0:6] = 0000101) allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOSC[n] register.

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOSA[n] and EMIOSB[n] return the values in register A2 and B1, respectively.

In order to allow coherent data, reading EMIOSA[n] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the EMIOSB[n] register. Reading EMIOSB[n] register forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 375 shows how the Unified Channel can be used for input period measurement.

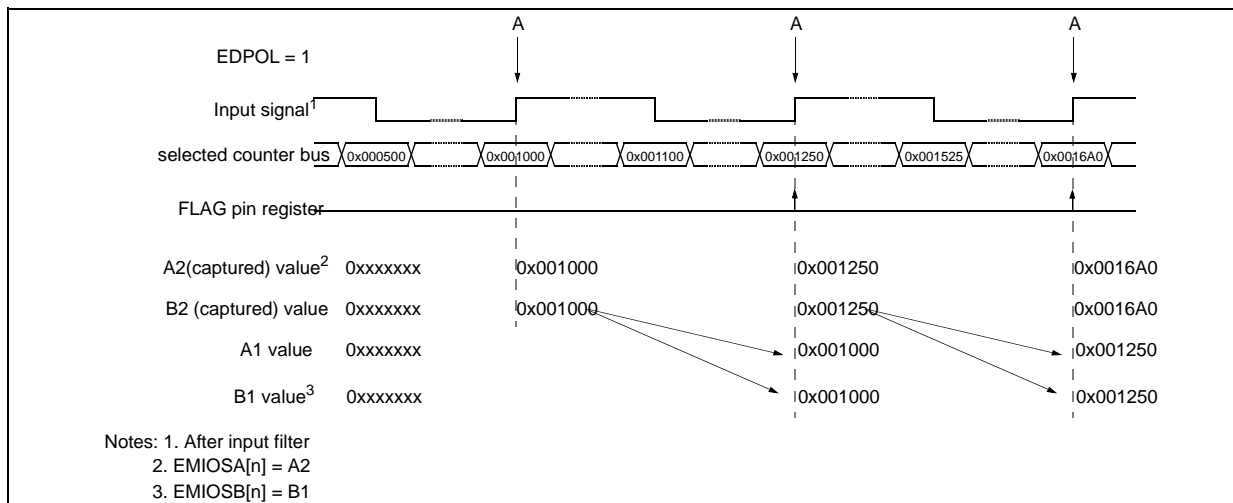


Figure 375. Input period measurement example

Figure 376 describes the A1 and B1 register updates when EMIOSA[n] and EMIOSB[n] read operations are performed. When EMIOSA[n] read occurs the content of A1 is transferred to B1 thus providing coherent data in A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOSB[n] is read. After EMIOSB[n] is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the Figure 376 example.

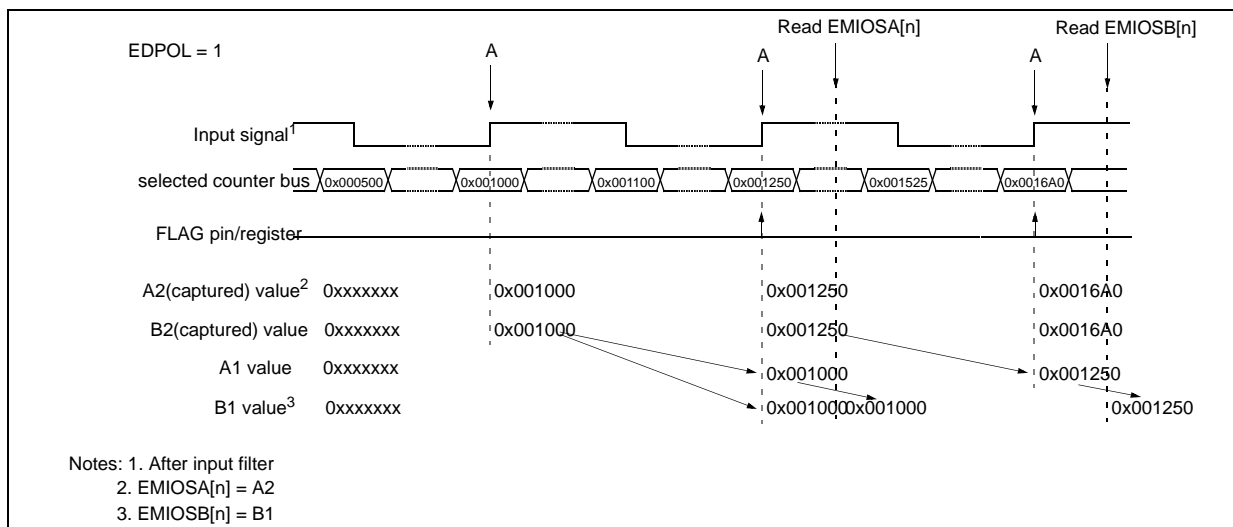


Figure 376. A1 and B1 updates at EMIOSA[n] and EMIOSB[n] reads

### 27.4.4.1.1.6 Double Action Output Compare (DAOC) mode

In the DAOC mode the leading and trailing edges of the variable pulse width output are generated by matches occurring on comparators A and B. There is no restriction concerning the order in which A and B matches occur.



When the DAOC mode is entered, coming out from GPIO mode both comparators are disabled and the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

Data written to A2 and B2 are transferred to A1 and B1, respectively, on the next system clock cycle if bit OU[n] of the EMIOSOUDIS register is cleared (see [Figure 379](#)). The transfer is blocked if bit OU[n] is set. Comparator A is enabled only after the transfer to A1 register occurs and is disabled on the next A match. Comparator B is enabled only after the transfer to B1 register occurs and is disabled on the next B match. Comparators A and B are enabled and disabled independently.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

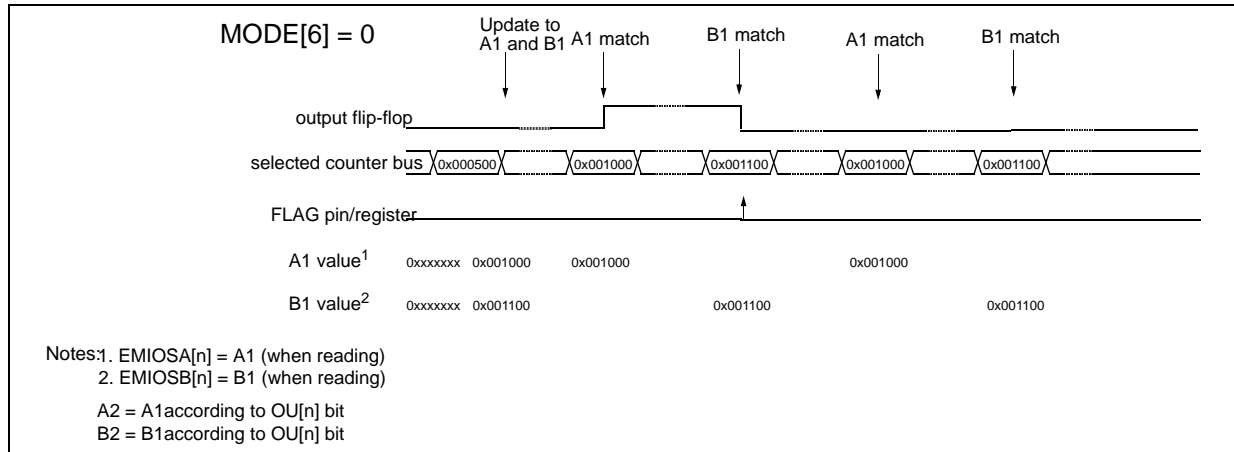
MODE[6] controls if the FLAG is set on both matches (MODE[0:6] = 0000111) or just on the B match (MODE[0:6] = 0000110). FLAG bit assertion depends on comparator enabling.

If subsequent enabled output compares occur on registers A1 and B1, pulses will continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. Note that the FLAG bit is not affected by these forced operations.

*Note: If both registers (A1 and B1) are loaded with the same value, the B match prevails concerning the output pin state (output flip-flop is set to the complement of EDPOL), the FLAG bit is set and both comparators are disabled.*

[Figure 377](#) and [Figure 378](#) show how the Unified Channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.



**Figure 377. Double action output compare with FLAG set on the second match**

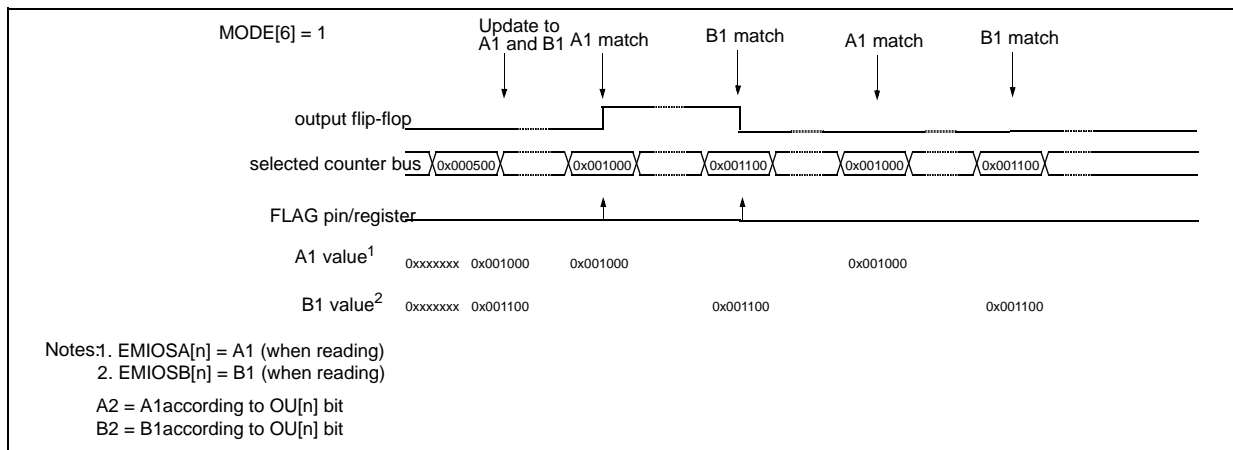


Figure 378. Double action output compare with FLAG set on both matches

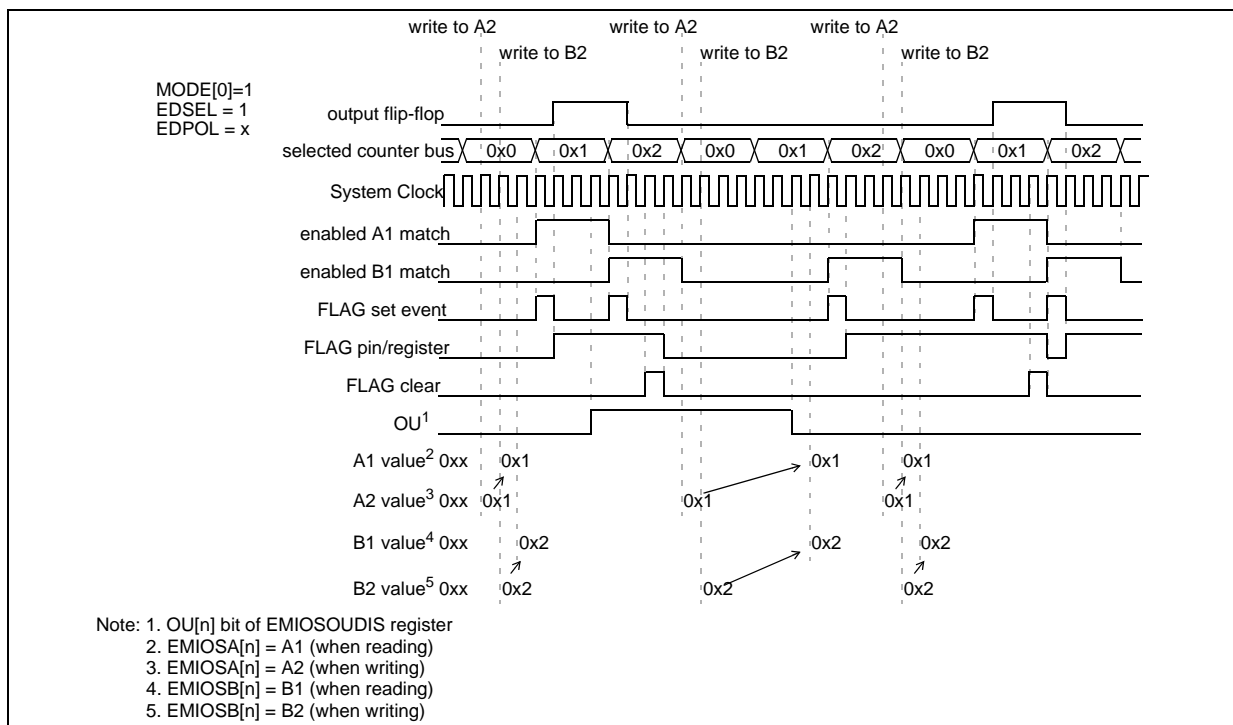


Figure 379. DAOC with transfer disabling example

27.4.4.1.1.7 Modulus Counter (MC) mode

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

Bit MODE[6] selects internal or external clock source when cleared or set, respectively. When external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL in the EMIOSC[n] register.

The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not accessible to the MCU. Bit MODE[4] selects up mode or up/down mode, when cleared or set, respectively.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter. The timing of those events varies according to the MC mode setup as follows:

- Internal counter clearing on match start (MODE[0:6] = 001000b)
  - External clock is selected if MODE[6] is set. In this case the internal counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. Note that by having the internal counter cleared as soon as the match occurs and incremented at the next input event a shorter zero count is generated. See [Figure 402](#) and [Figure 403](#).
  - Internal clock source is selected if MODE[6] is cleared. In this case the counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. At the next prescaler tick after the match the internal counter remains at zero and only resumes counting on the following tick. See [Figure 402](#) and [Figure 404](#).
- Internal counter clearing on match end (MODE[0:6] = 001001b)
  - External clock is selected if MODE[6] is set. In this case the internal counter clears when the match signal is asserted and the input event occurs. The channel FLAG is set at the same time the counter is cleared. See [Figure 402](#) and [Figure 405](#).
  - Internal clock source is selected if MODE[6] is cleared. In this case the internal counter clears when the match signal is asserted and the prescaler tick occurs. The channel FLAG is set at the same time the counter is cleared. See [Figure 402](#) and [Figure 405](#).

*Note:* If the internal clock source is selected and the prescaler of the internal counter is set to '1', the MC mode behaves the same way even in Clear on Match Start or Clear on Match End submodes.

When in up/down count mode (MODE[0:6] = 00101bb), a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[5] bit is set.

Only values different than 0x0 must be written at A register. Loading 0x0 leads to unpredictable results.

Updates on A register or counter in MC mode may cause loss of match in the current cycle if the transfer occurs near the match. In this case, the counter may rollover and resume operation in the next cycle.

Register B2 has no effect in MC mode. Nevertheless, register B2 can be accessed for reads and writes by addressing EMIOSB.

[Figure 380](#) and [Figure 381](#) show how the Unified Channel can be used as modulus counter in up mode and up/down mode, respectively.

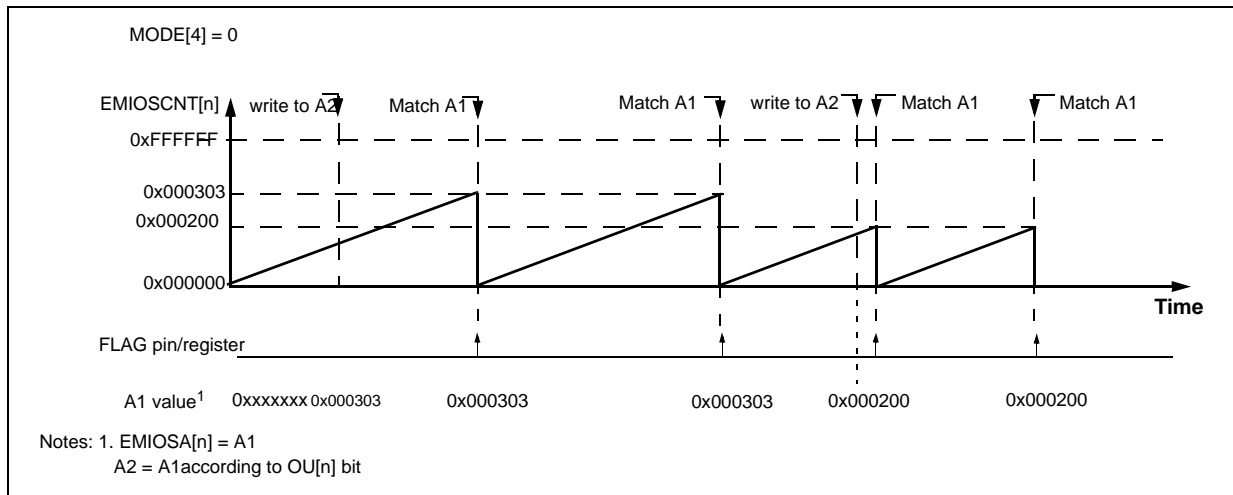


Figure 380. Modulus Counter Up mode example

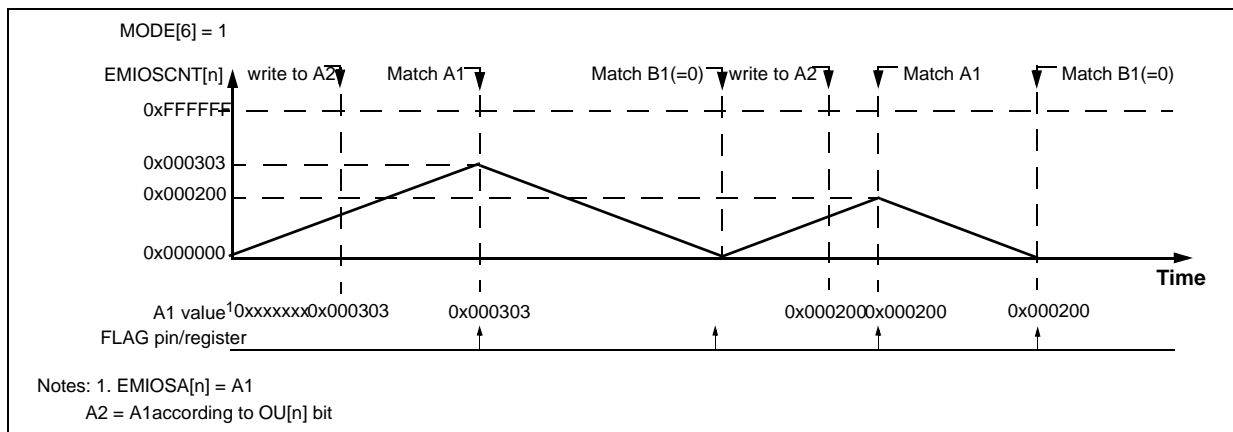


Figure 381. Modulus Counter Up/Down mode example

27.4.4.1.1.8 Modulus Counter Buffered (MCB) mode

The MCB mode provides a time base which can be shared with other channels through the internal counter buses. Register A1 is double buffered thus allowing smooth transitions between cycles when changing A2 register value on the fly. A1 register is updated at the cycle boundary, which is defined as when the internal counter transitions to 0x1.

The internal counter values operates within a range from 0x1 up to register A1 value. If when entering MCB mode coming out from GPIO mode the internal counter value is not within that range then the A match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xFFFF for a 16-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal MCB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to A1 register value range when the MCB mode is entered.

Bit MODE[6] selects internal clock source if cleared or external if set. When external clock is selected the input channel pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the EMIOSC[n] channel register.

When entering in MCB mode, if up counter is selected by  $MODE[4] = 0$  ( $MODE[0:6] = 101000b$ ), the internal counter starts counting from its current value to up direction until A1 match occurs. The internal counter is set to 0x1 when its value matches A1 value and a clock tick occurs (either prescaled clock or input pin event).

If up/down counter is selected by setting  $MODE[4] = 1$ , the counter changes direction at A1 match and counts down until it reaches the value 0x1. After it has reached 0x1 it is set to count in up direction again. B1 register is used to generate a match in order to set the internal counter in up-count direction if up/down mode is selected. Register B1 cannot be changed while this mode is selected.

Note that differently from the MC mode, the MCB mode counts between 0x1 and A1 register value. Only values greater than 0x1 must be written at A1 register. Loading values other than those leads to unpredictable results. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode the period is defined by the expression:  $(2 * A1) - 2$ .

Figure 382 describes the counter cycle for several A1 values. Register A1 is loaded with A2 register value at the cycle boundary. Thus any value written to A2 register within cycle n will be updated to A1 at the next cycle boundary and therefore will be used on cycle n+1. The cycle boundary between cycle n and cycle n+1 is defined as when the internal counter transitions from A1 value in cycle n to 0x1 in cycle n+1. Note that the FLAG is generated at the cycle boundary and has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

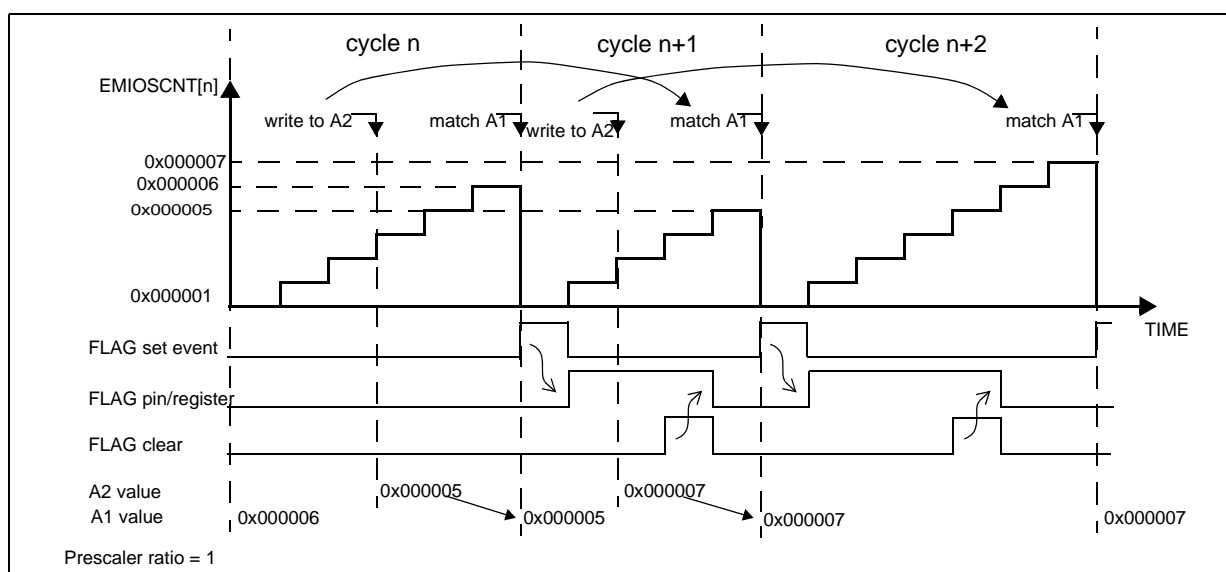
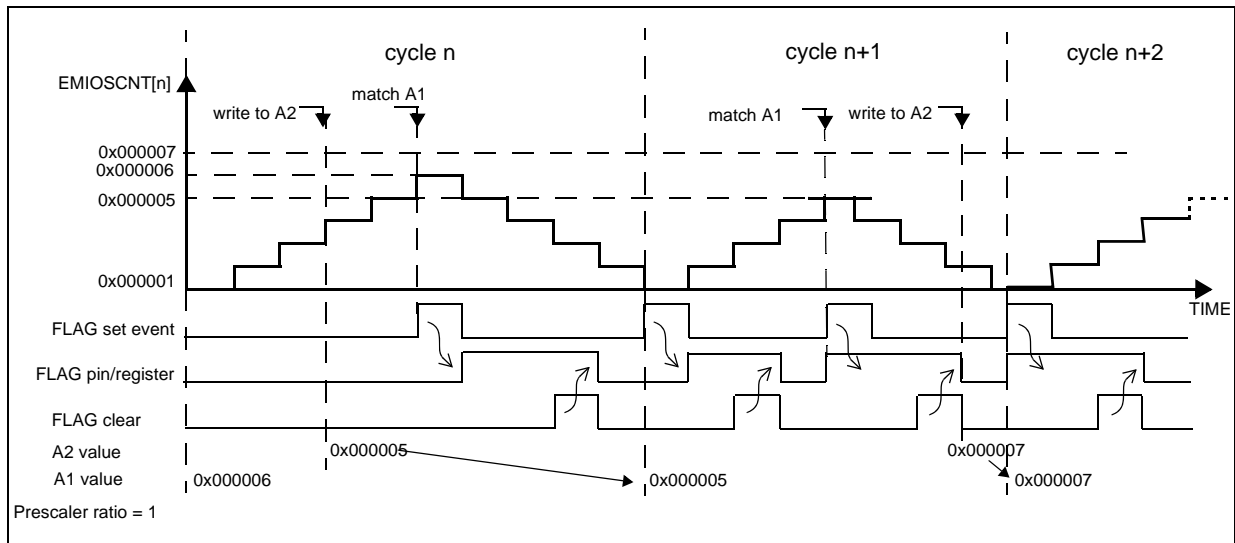


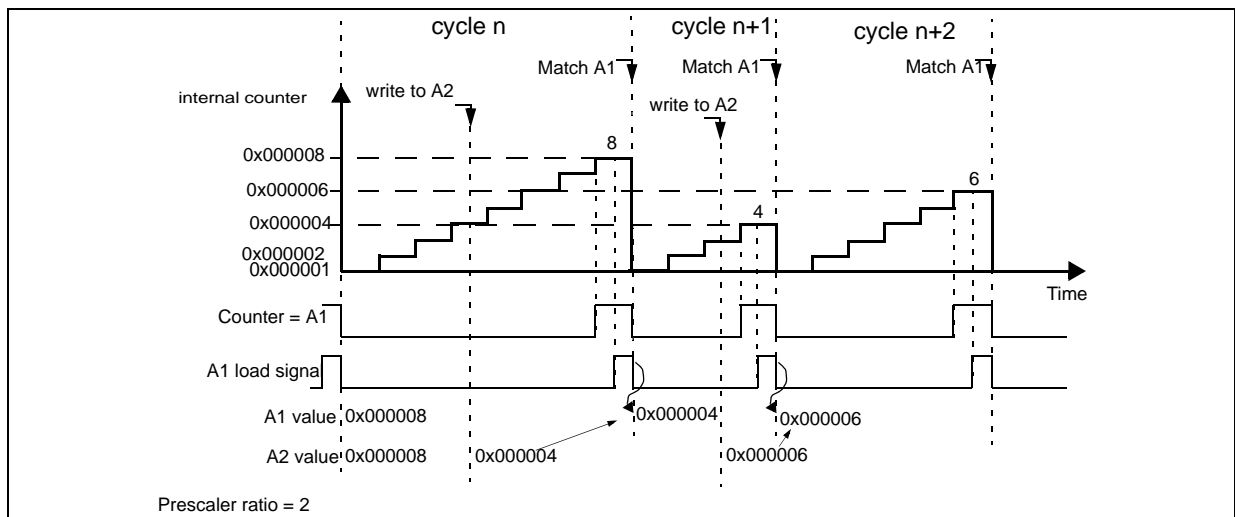
Figure 382. Modulus Counter Buffered (MCB) Up Count mode

Figure 383 describes the MCB in up/down counter mode ( $MODE[0:6] = 10101bb$ ). A1 register is updated at the cycle boundary. If A2 is written in cycle n, this new value will be used in cycle n+1 for A1 match. Flags are generated only at A1 match start if  $MODE[5]$  is 0. If  $MODE[5]$  is set to 1 flags are also generated at the cycle boundary.



**Figure 383. Modulus Counter Buffered (MCB) Up/Down mode**

Figure 384 describes in more detail the A1 register update process in up counter mode. The A1 load signal is generated at the last system clock period of a counter cycle. Thus, A1 is updated with A2 value at the same time that the counter (EMIOSCNT[n]) is loaded with 0x1. The load signal pulse has the duration of one system clock period. If A2 is written within cycle n its value is available at A1 at the first clock of cycle n+1 and the new value is used for match at cycle n+1. The update disable bits OU[n] of EMIOSOUDIS register can be used to control the update of this register, thus allowing to delay the A1 register update for synchronization purposes.



**Figure 384. MCB Mode A1 Register Update in Up Counter mode**

Figure 385 describes the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle n in order to be used in cycle n+1. Thus A1 receives this new value at the next cycle boundary. Note that the update disable bits OU[n] of EMIOSOUDIS register can be used to disable the update of A1 register.

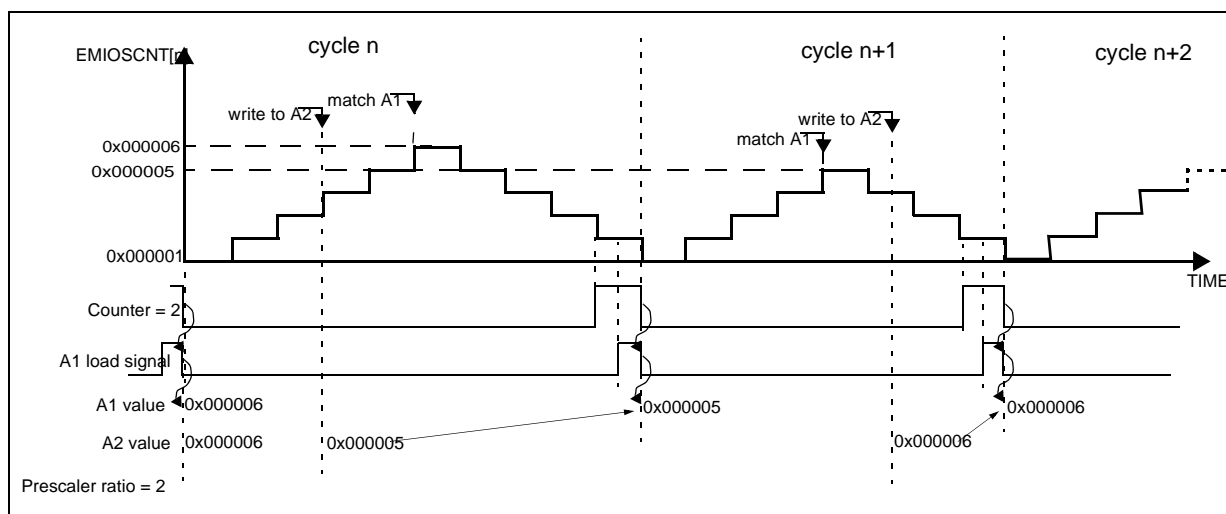


Figure 385. MCB Mode A1 Register Update in Up/Down Counter mode

### 27.4.4.2 Output Pulse Width and Frequency Modulation Buffered (OPWFMB) mode

This mode (MODE[0:6] = 10110b0) provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to allow smooth signal generation when changing the registers values on the fly. 0% and 100% duty cycles are supported.

At OPWFMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOSC[n] register.

If when entering OPWFMB mode coming out from GPIO mode the internal counter value is not within that range then the B match will not occur causing the channel internal counter to wrap at the maximum counter value which is 0xFFFF for a 16-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal OPWFMB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to B1 register value range when the OPWFMB mode is entered.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 0x1, thus restarting the counter cycle.

Only values greater than 0x1 are allowed to be written to B1 register. Loading values other than those leads to unpredictable results. If you want to configure the module for OPWFMB mode, ensure that the B1 register is modified before the mode is set.

*Note: The standard way to create a counter bus is by configuring channel 23 (global) or any of channels 0, 8, 16 or 24 (local groups) in MC or MCB mode. These counter modes will generate an internal count that can be read by other channels when selecting the proper counter driver. However, OPWFM and OPWFMB modes use the channel internal counter for their inner processes, and these counts may be used by other channels in the same way as in MC and MCB ones. This enables the user for example, to configure a OPWFMB mode in channel 0 and*

OPWMB in channels 1 to 7 so that the whole set could generate aligned PWM's based on the same counter bus generated by channel 0 (and without wasting CH0 as MCB).

Figure 386 describes the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. Note that the output pin transition occurs when the A1 or B1 match signal is deasserted which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x4 the output pin transitions 4 counter periods after the cycle had started, plus one system clock cycle. Note that in the example shown in Figure 386 the internal counter prescaler has a ratio of two.

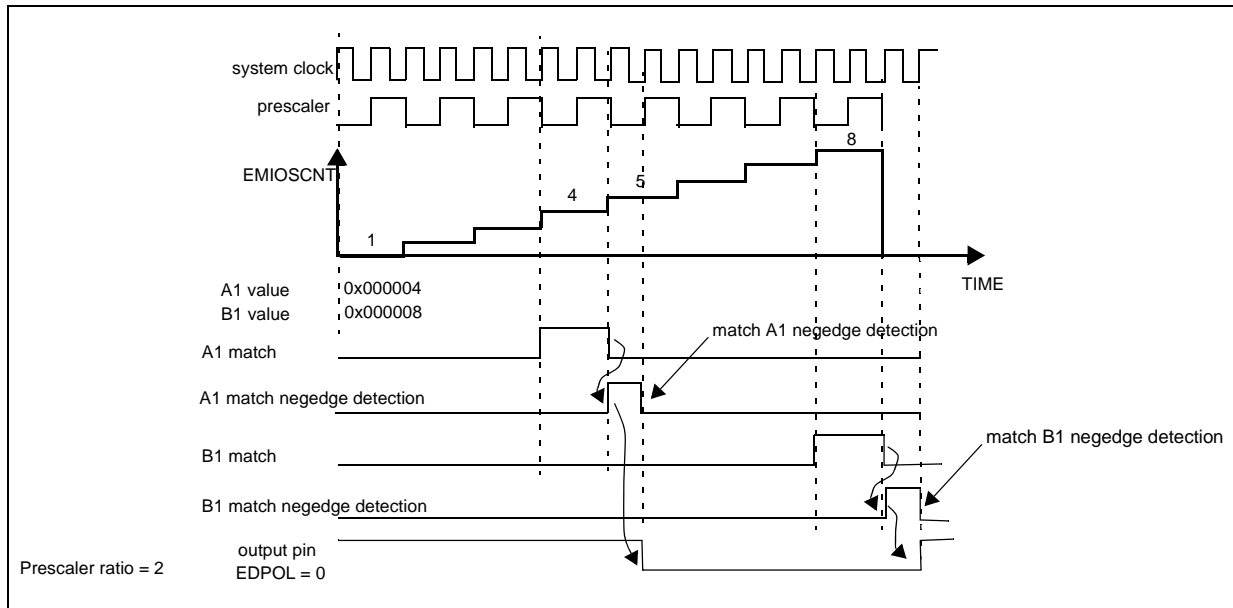
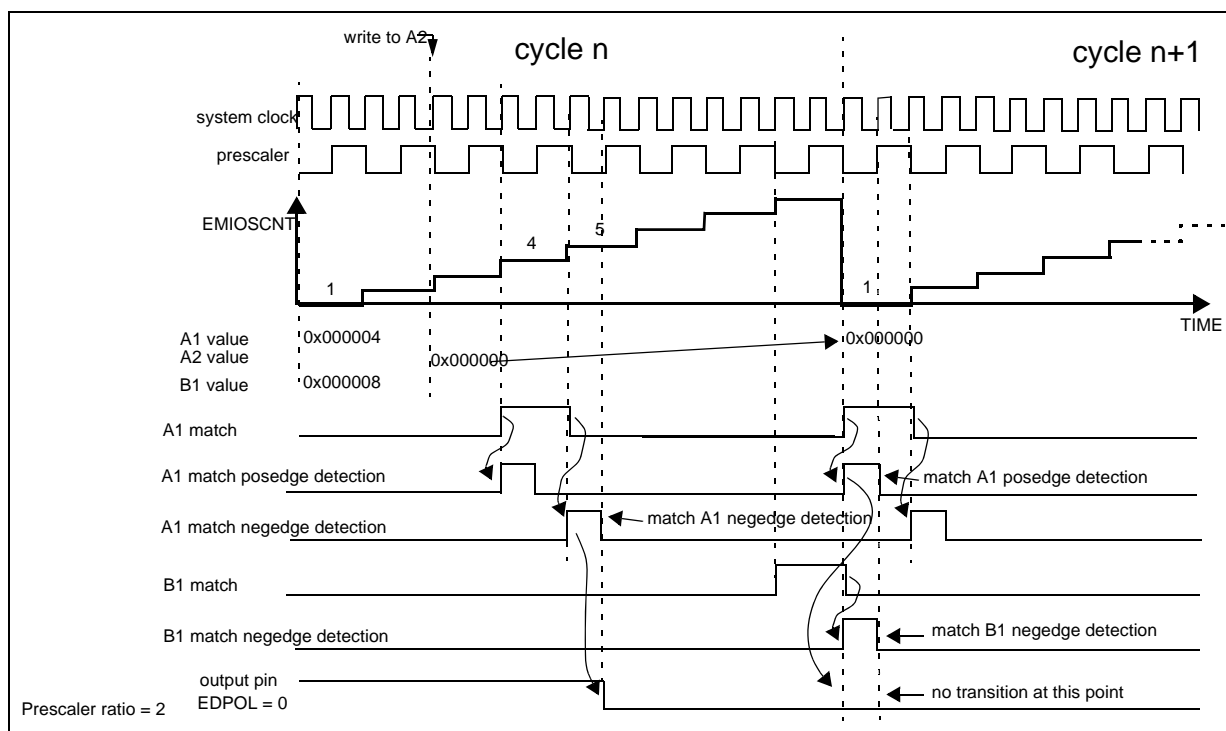


Figure 386. OPWFMB A1 and B1 match to Output Register Delay

Figure 387 describes the generated output signal if A1 is set to 0x0. Since the counter does not reach zero in this mode, the channel internal logic infers a match as if A1 = 0x1 with the difference that in this case, the posedge of the match signal is used to trigger the output pin transition instead of the negedge used when A1 = 0x1. Note that A1 posedge match signal from cycle  $n+1$  occurs at the same time as B1 negedge match signal from cycle  $n$ . This allows to use the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

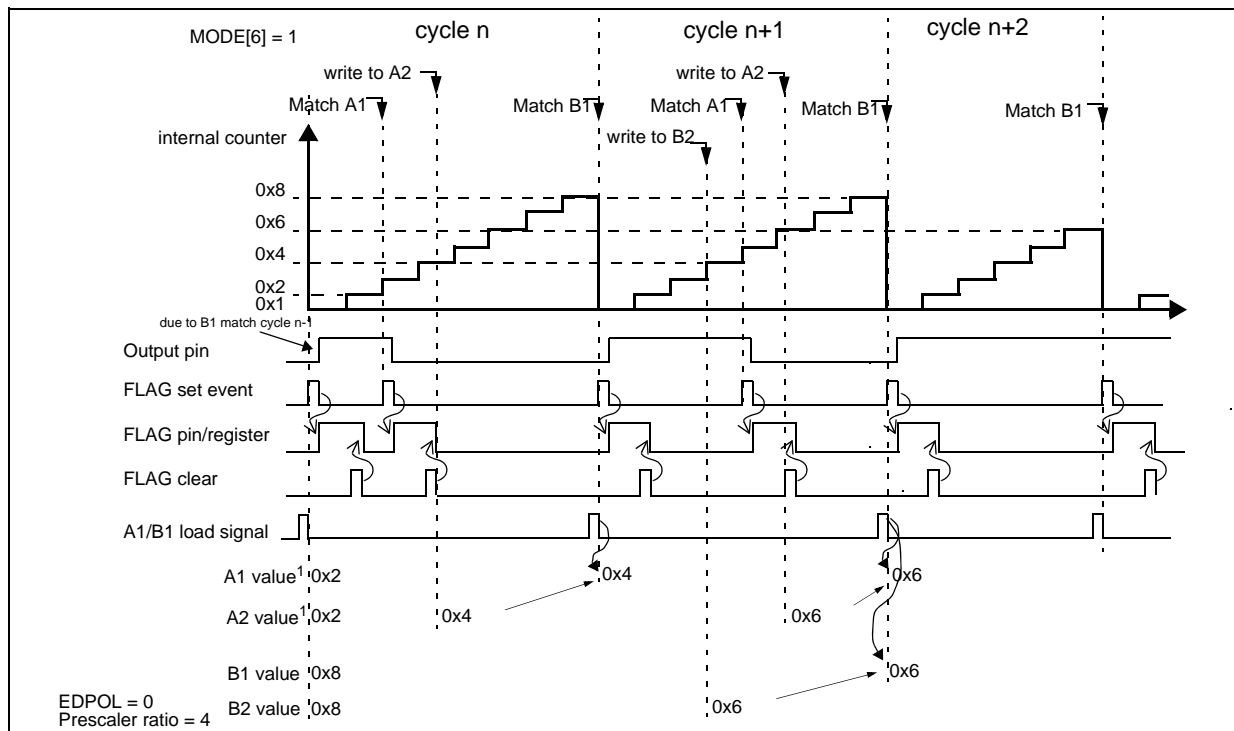




**Figure 387. OPWFMB Mode with A1 = 0 (0% duty cycle)**

[Figure 388](#) describes the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal which is generated at the last system clock period of a counter cycle. Thus, A1 and B1 are updated respectively with A2 and B2 values at the same time that the counter (EMIOSCNT[n]) is loaded with 0x1. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock period. If A2 and B2 are written within cycle **n** their values are available at A1 and B1, respectively, at the first clock of cycle **n+1** and the new values are used for matches at cycle **n+1**. The update disable bits OU[n] of EMIOSOUDIS register can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

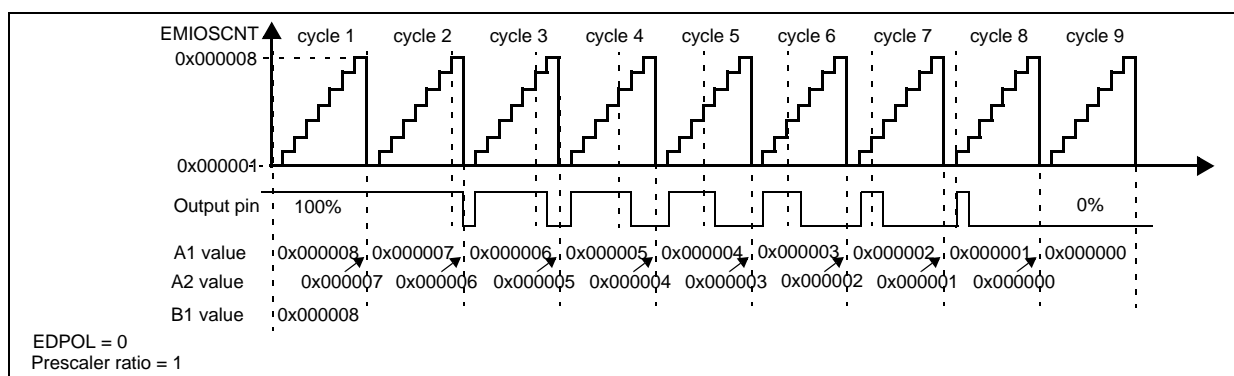
In [Figure 388](#) it is assumed that both the channel and global prescalers are set to 0x1 (each divide ratio is two), meaning that the channel internal counter transitions at every four system clock cycles. FLAGS can be generated only on B1 matches when MODE[5] is cleared, or on both A1 and B1 matches when MODE[5] is set. Since B1 flag occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle **n** were loaded to A1 or B1, respectively, thus generating matches in cycle **n+1**. Note that the FLAG has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.



**Figure 388. OPWFMB A1 and B1 registers update and flags**

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similarly to a B1 match FORCMB sets the internal counter to 0x1. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 389 describes the generation of 100% and 0% duty cycle signals. It is assumed EDPOL = 0 and the resultant prescaler value is 1. Initially A1 = 0x8 and B1 = 0x8. In this case, B1 match has precedence over A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.



**Figure 389. OPWFMB mode from 100% to 0% duty cycle**

A 0% duty cycle signal is generated if A1 = 0x0 as shown in Figure 389 cycle 9. In this case B1 = 0x8 match from cycle 8 occurs at the same time as the A1 = 0x0 match from cycle 9.

Please, refer to [Figure 387](#) for a description of the A1 and B1 match generation. In this case A1 match has precedence over B1 match and the output signal transitions to EDPOL.

#### 27.4.4.2.0.1 Center Aligned Output PWM Buffered with Dead-Time (OPWMCB) mode

This operation mode generates a center aligned PWM with dead time insertion to the leading (MODE[0:6] = 10111b1) or trailing edge (MODE[0:6] = 10111b0). A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 registers values on the fly.

Bits BSL[0:1] select the time base. The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB Up/Down mode, as shown in [Figure 383](#). It is recommended to start the MCB channel time base after the OPWMCB mode is entered in order to avoid missing A matches at the very first duty cycle.

Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base.

Register B1 contains the dead time value and is compared against the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Bit Mode[6] selects between trailing and leading dead time insertion, respectively.

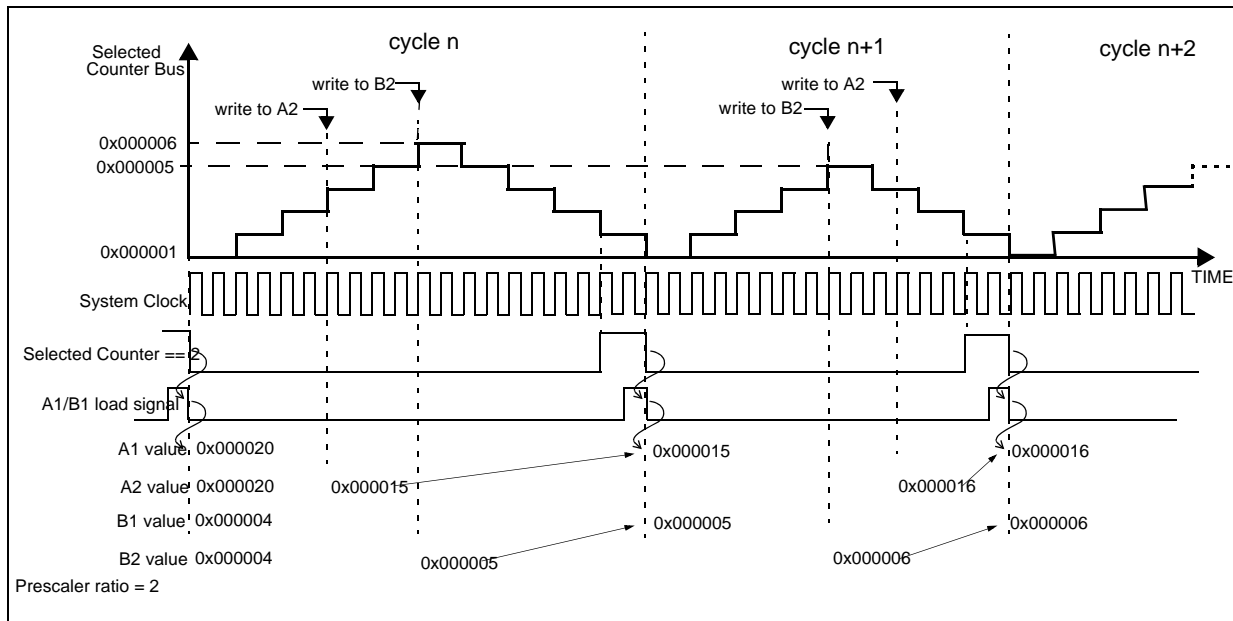
*Note:* *The internal counter runs in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio.*

When OPWMCB mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

The following basic steps summarize proper OPWMCB startup, assuming the channels are initially in GPIO mode:

1. *[global]* Disable Global Prescaler;
2. *[MCB channel]* Disable Channel Prescaler;
3. *[MCB channel]* Write 0x1 at internal counter;
4. *[MCB channel]* Set A register;
5. *[MCB channel]* Set channel to MCB Up mode;
6. *[MCB channel]* Set prescaler ratio;
7. *[MCB channel]* Enable Channel Prescaler;
8. *[OPWMCB channel]* Disable Channel Prescaler;
9. *[OPWMCB channel]* Set A register;
10. *[OPWMCB channel]* Set B register;
11. *[OPWMCB channel]* Select time base input through BSL[1:0] bits;
12. *[OPWMCB channel]* Enter OPWMCB mode;
13. *[OPWMCB channel]* Set prescaler ratio;
14. *[OPWMCB channel]* Enable Channel Prescaler;
15. *[global]* Enable Global Prescaler.

[Figure 390](#) describes the load of A1 and B1 registers which occurs when the selected counter bus transitions from 0x2 to 0x1. This event defines the cycle boundary. Note that values written to A2 or B2 within cycle **n** are loaded into A1 or B1 registers, respectively, and used to generate matches in cycle **n+1**.

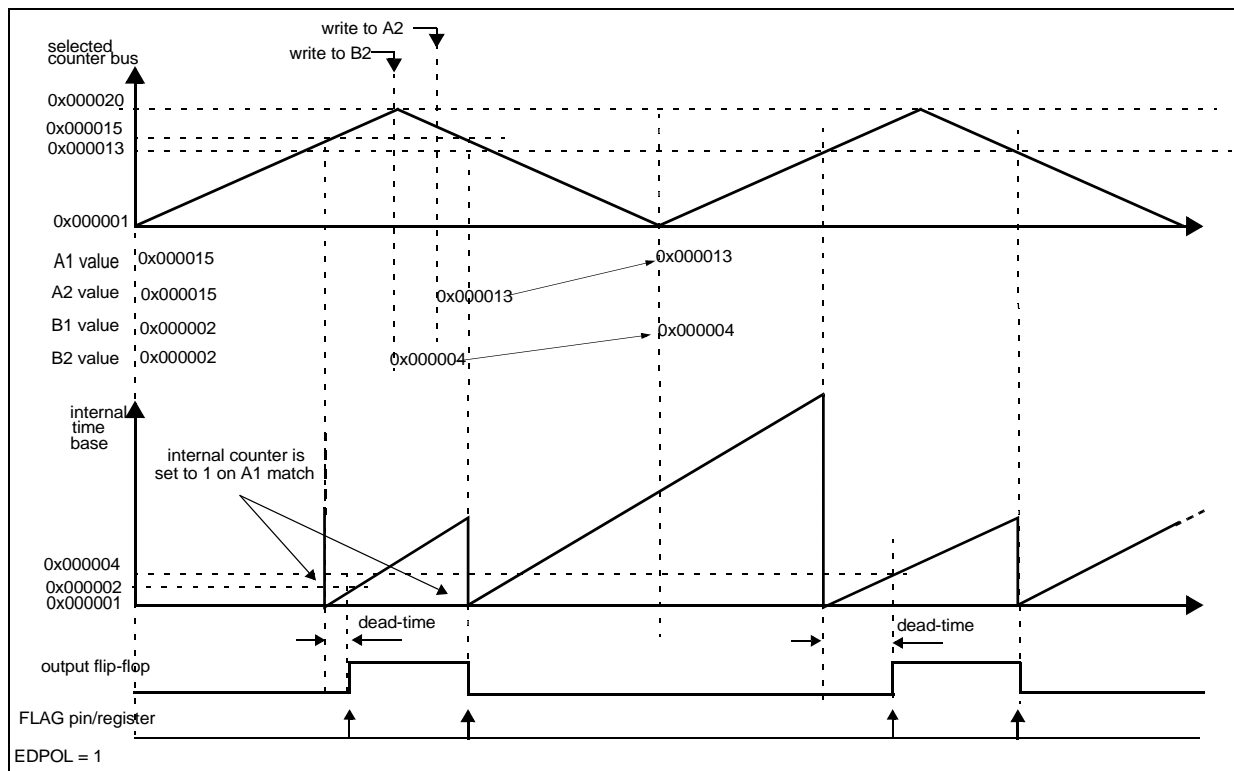


**Figure 390. OPWMCB A1 and B1 registers load**

Bit OU[n] of the EMIOSOUDIS register can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Note that using the update disable bit A1 and B1 registers can be updated at the same counter cycle thus allowing to change both registers at the same time.

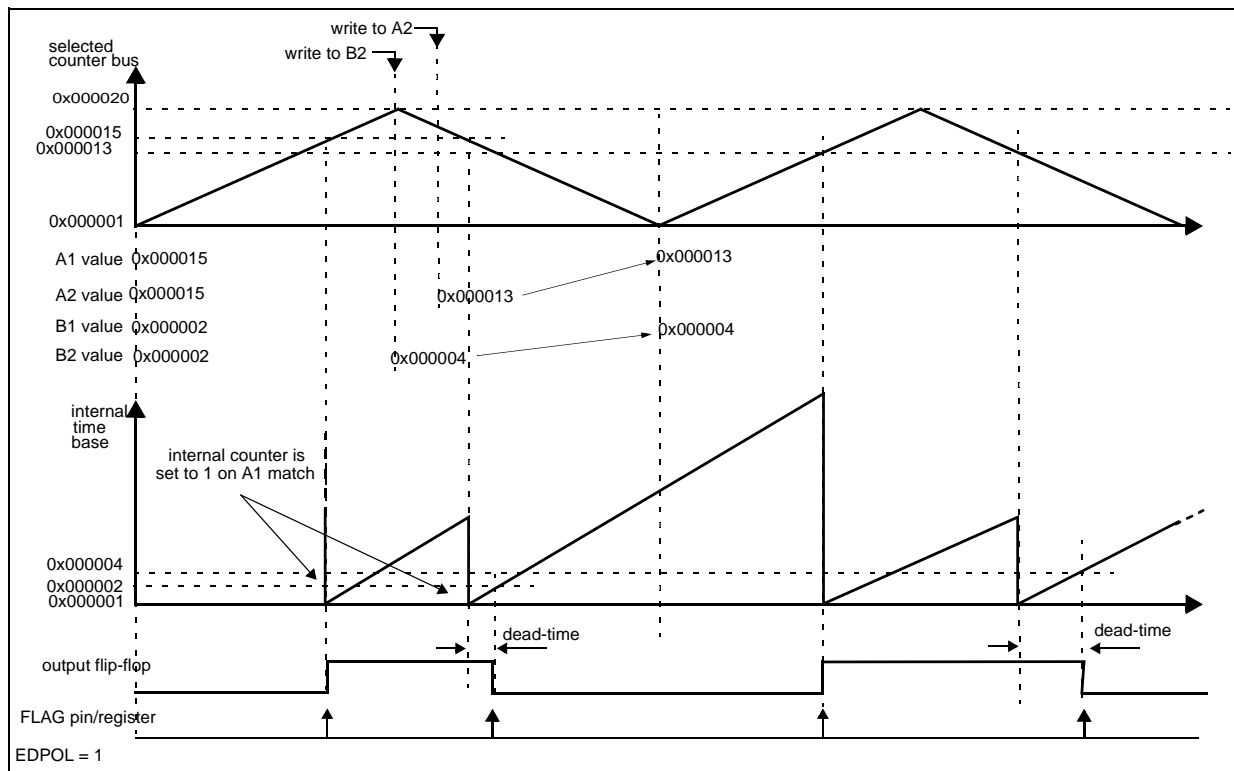
In this mode A1 matches always sets the internal counter to 0x1. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x1. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. The internal counter should not reach 0x0 as consequence of a rollover. In order to avoid it the user should not write to the EMIOSB register a value greater than twice the difference between external count up limit and EMIOSA value.

[Figure 391](#) shows two cycles of a Center Aligned PWM signal. Note that both A1 and B1 register values are changing within the same cycle which allows to vary at the same time the duty cycle and dead time values.



**Figure 391. OPWMCB with lead dead time insertion**

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x1. In the second match between register A1 and the selected time base, the internal counter is set to 0x1 and B1 matches are enabled. When the match between register B1 and the selected time base occurs the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.



**Figure 392. OPWMCB with trail dead time insertion**

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

*Note:* In OPWMCB mode, FORCMA and FORCMB do not have the same behavior as a regular match. Instead, they force the output flip-flop to constant value which depends upon the selected dead time insertion mode, lead or trail, and the value of the EDPOL bit.

FORCMA has different behaviors depending upon the selected dead time insertion mode, lead or trail. In lead dead time insertion FORCMA force a transition in the output flip-flop to the opposite of EDPOL. In trail dead time insertion the output flip-flop is forced to the value of EDPOL bit.

If bit FORCMB is set, the output flip-flop value depends upon the selected dead time insertion mode. In lead dead time insertion FORCMB forces the output flip-flop to transition to EDPOL bit value. In trail dead time insertion the output flip-flop is forced to the opposite of EDPOL bit value.

*Note:* FORCMA bit set does not set the internal time-base to 0x1 as a regular A1 match.

The FLAG bit is not set either in case of a FORCMA or FORCMB or even if both forces are issued at the same time.

*Note:* FORCMA and FORCMB have the same behavior even in Freeze or normal mode regarding the output pin transition.

When FORCMA is issued along with FORCMB the output flip-flop is set to the opposite of EDPOL bit value. This is equivalent of saying that.FORCMA has precedence over FORCMB

when lead dead time insertion is selected and FORCMB has precedence over FORCMA when trail dead time insertion is selected.

Duty cycle from 0% to 100% can be generated by setting appropriate values to A1 and B1 registers relatively to the period of the external time base. Setting A1 = 1 generates a 100% duty cycle waveform. Assuming EDPOL is set to '1' and OPWMCB mode with trail dead time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1). If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced, only if the pin starts the current cycle in the opposite of EDPOL value. In case of 100% duty cycle, the transition from EDPOL to the opposite of EDPOL may be obtained by forcing pin, using FORCMA or FORCMB, or both.

*Note:* If A1 is set to 0x1 at OPWMCB entry the 100% duty cycle may not be obtained in the very first PWM cycle due to the pin condition at mode entry.

Only values different than 0x0 are allowed to be written to A1 register. If 0x0 is loaded to A1 the results are unpredictable.

*Note:* A special case occurs when A1 is set to (external counter bus period)/2, which is the maximum value of the external counter. In this case the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trail dead time insertion B1 match from cycle n could eventually cross the cycle boundary and occur in cycle n+1. In this case B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle n+1 are not affected by the late B1 matches from cycle n.

Figure 393 shows a 100% duty cycle output signal generated by setting A1 = 4 and B1 = 3. In this case the trailing edge is positioned at the boundary of cycle n+1, which is actually considered to belong to cycle n+2 and therefore does not cause the output flip-flop to transition.

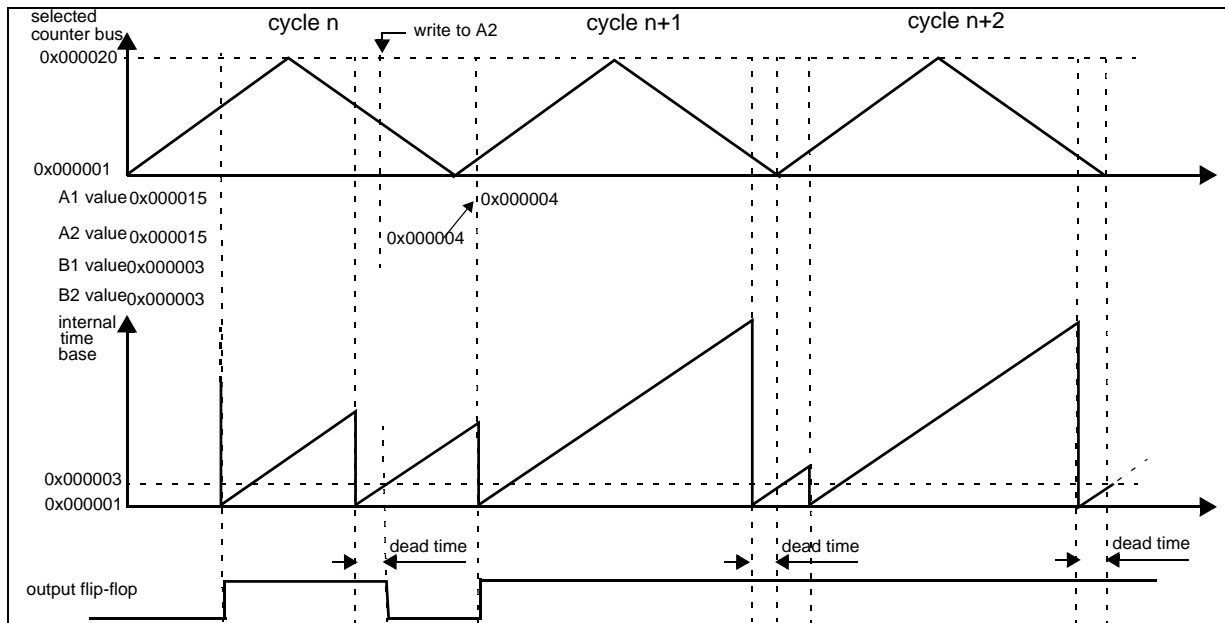


Figure 393. OPWMCB with 100 % Duty Cycle (A1 = 4 and B1 = 3)

It is important to notice that, such as in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal which compares the selected time base with A1 or B1 register values. Please refer to [Figure 386](#) which describes the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

### 27.4.4.3 Output Pulse Width Modulation Buffered (OPWMB) Mode

OPWMB mode (MODE[0:6] = 11000b0) is used to generate pulses with programmable leading and trailing edge placement. An external counter driven in MCB Up mode must be selected from one of the counter buses. A1 register value defines the first edge and B1 the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Please refer to [Figure 388](#) for more information about A1 and B1 registers update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or in both A1 and B1 matches, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. FLAG bit is not set by the FORCMA and FORCMB operations.

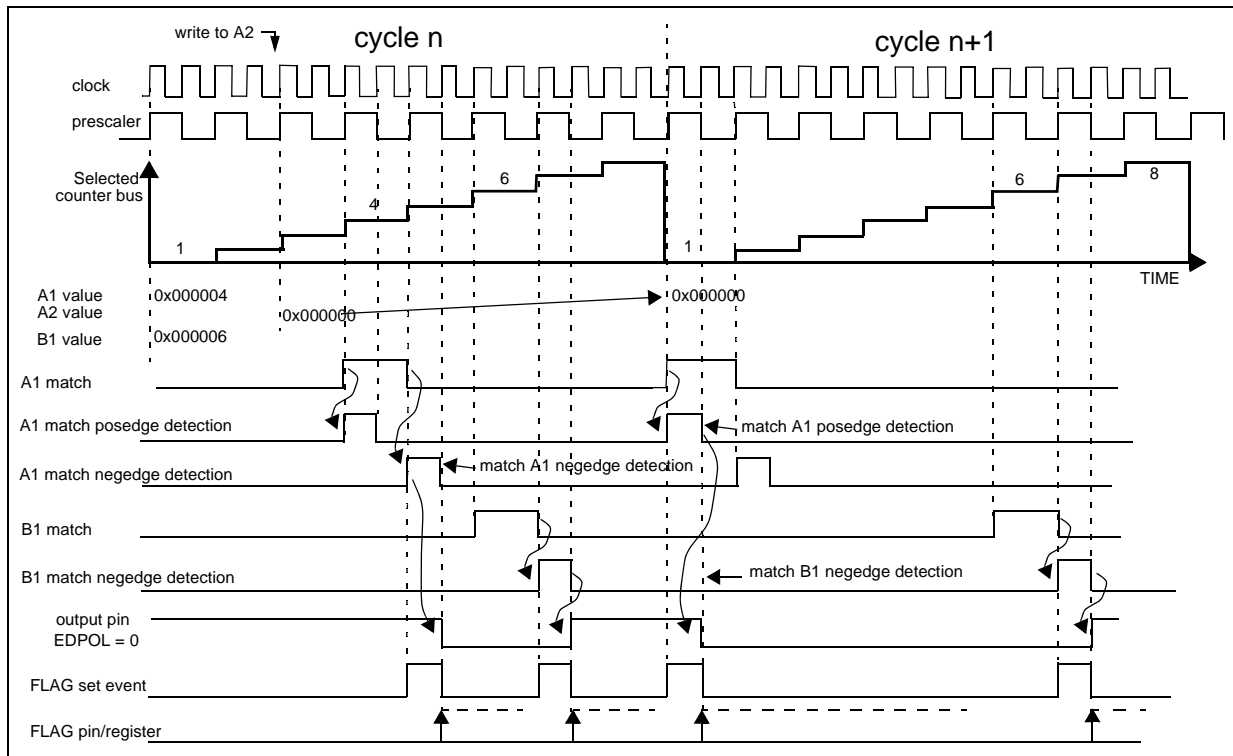
At OPWMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOSC[n] register.

Some rules applicable to the OPWMB mode are:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1 = 0 match from cycle **n** has precedence over B1 match from cycle **n-1**
- A1 matches are masked out if they occur after B1 match within the same cycle
- Any value written to A2 or B2 on cycle **n** is loaded to A1 and B1 registers at the following cycle boundary (assuming OU[n] bit of EMIOSOUDIS register is not asserted). Thus the new values will be used for A1 and B1 matches in cycle **n+1**

[Figure 394](#) describes the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example EDPOL is set to '0'.





**Figure 394. OPWMB mode matches and flags**

Note that the output pin transitions are based on the negedges of the A1 and B1 match signals. [Figure 394](#) shows in cycle **n+1** the value of A1 register being set to '0'. In this case the match posedge is used instead of the negedge to transition the output flip-flop.

[Figure 395](#) describes the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1 = 0x8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

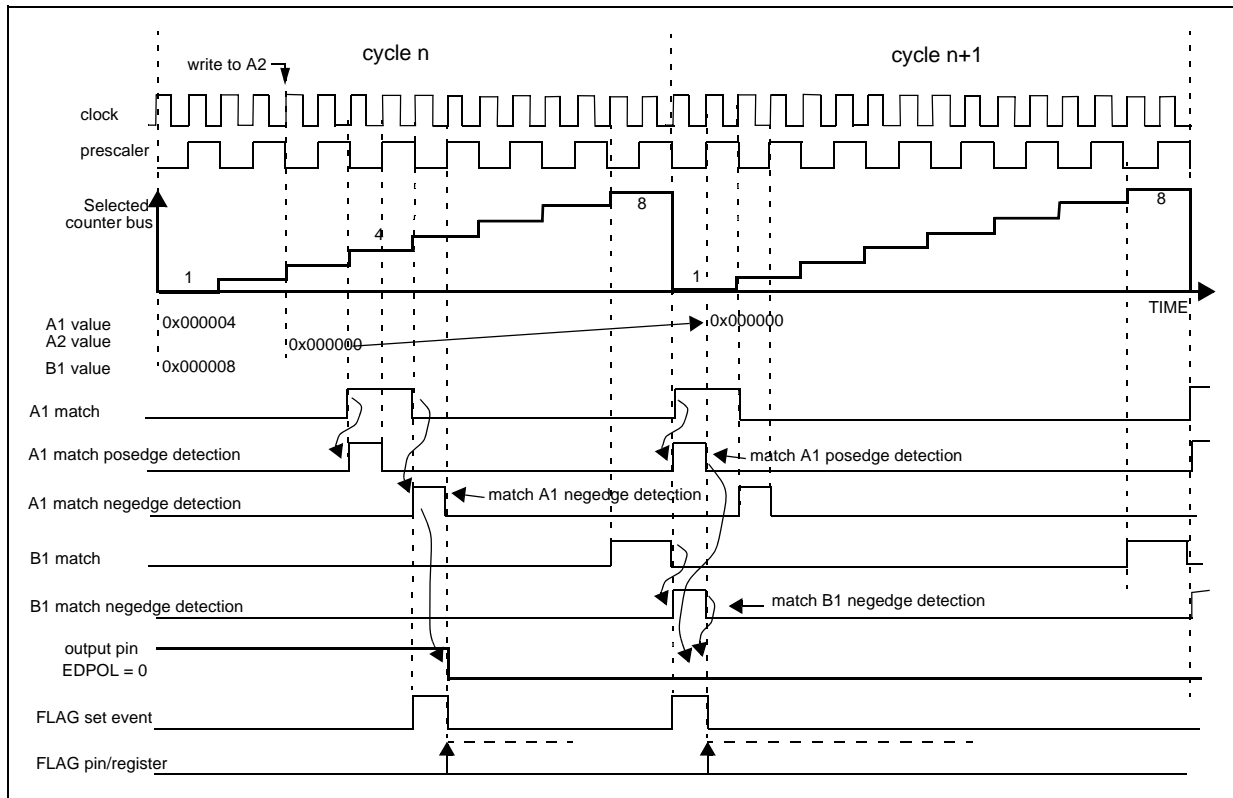


Figure 395. OPWMB mode with 0% duty cycle

Figure 396 shows a waveform changing from 100% to 0% duty cycle. EDPOL in this case is zero. In this example B1 is programmed to the same value as the period of the external selected time base.

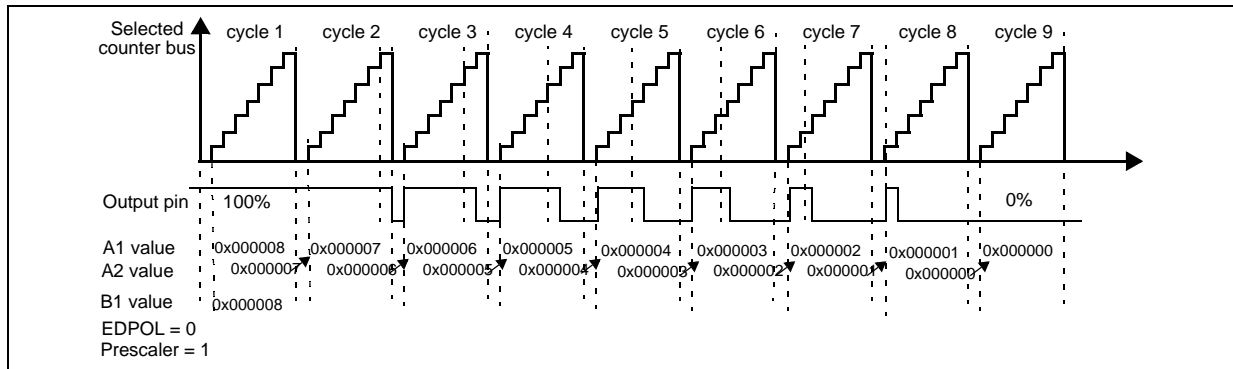


Figure 396. OPWMB mode from 100% to 0% duty cycle

In Figure 396 if B1 is set to a value lower than 0x8 it is not possible to achieve 0% duty cycle by only changing A1 register value. Since B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. Note also that if B1 is set to 0x9, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

#### 27.4.4.4 Output Pulse Width Modulation with Trigger (OPWMT) mode

OPWMT mode (MODE[0:6] = 0100110) is intended to support the generation of pulse width modulation signals where the period is not modified while the signal is being output, but where the duty cycle will be varied and must not create glitches. The mode is intended to be used in conjunction with other channels executing in the same mode and sharing a common timebase. It will support each channel with a fixed PWM leading edge position with respect to the other channels and the ability to generate a trigger signal at any point in the period that can be output from the module to initiate activity in other parts of the device such as starting ADC conversions.

An external counter driven in either MC Up or MCB Up mode must be selected from one of the counter buses.

Register A1 defines the leading edge of the PWM output pulse and as such the beginning of the PWM's period. This makes it possible to insure that the leading edge of multiple channels in OPWMT mode can occur at a specific time with respect to the other channels when using a shared timebase. This can allow the introduction of a fixed offset for each channel which can be particularly useful in the generation of lighting PWM control signals where it is desirable that edges are not coincident with each other to help eliminate noise generation. The value of register A1 represents the shift of the PWM channel with respect to the selected timebase. A1 can be configured with any value within the range of the selected time base. Note that registers loaded with 0x0 will not produce matches if the timebase is driven by a channel in MCB mode.

A1 is not buffered as the shift of a PWM channel must not be modified while the PWM signal is being generated. In case A1 is modified it is immediately updated and one PWM pulse could be lost.

EMIOSB[n] address gives access to B2 register for write and B1 register for read. Register B1 defines the trailing edge of the PWM output pulse and as such the duty cycle of the PWM signal. To synchronize B1 update with the PWM signal and so ensure a correct output pulse generation the transfer from B2 to B1 is done at every match of register A1.

EMIOSOUDIS register affects transfers between B2 and B1 only.

In order to account for the shift in the leading edge of the waveform defined by register A1 it will be necessary that the trailing edge, held in register B1, can roll over into the next period. This means that a match against the B1 register should not have to be qualified by a match in the A1 register. The impact of this would mean that incorrectly setting register B1 to a value less than register A1 will result in the output being held over a cycle boundary until the B1 value is encountered.

This mode provides a buffered update of the trailing edge by updating register B1 with register B2 contents only at a match of register A1.

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A1 occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

Note that the output pin and flag transitions are based on the posedges of the A1, B1 and A2 match signals. Please, refer to [Figure 394](#) at [Section 27.4.4.3: Output Pulse Width Modulation Buffered \(OPWMB\) Mode](#) for details on match posedge.

Register A2 defines the generation of a trigger event within the PWM period and A2 should be configured with any value within the range of the selected time base, otherwise no trigger will be generated. A match on the comparator will generate the FLAG signal but it has no

effect on the PWM output signal generation. The typical setup to obtain a trigger with FLAG is to enable DMA and to drive the channel's ipd\_done input high.

A2 is not buffered and therefore its update is immediate. If the channel is running when a change is made this could cause either the loss of one trigger event or the generation of two trigger events within the same period. Register A2 can be accessed by reading or writing the eMIOS UC Alternate A Register (EMIOSALTA) at UC[n] base address +0x14.

FLAG signal is set only at match on the comparator with A2. A match on the comparator with A1 or B1 or B2 has no effect on FLAG.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Any FORCMA and/or FORCMB has priority over any simultaneous match regarding to output pin transitions. Note that the load of B2 content on B1 register at an A match is not inhibited due to a simultaneous FORCMA/FORCMB assertion. If both FORCMA and FORCMB are asserted simultaneously the output pin goes to the opposite of EDPOL value such as if A1 and B1 registers had the same value. FORCMA assertion causes the transfer from register B2 to B1 such as a regular A match, regardless of FORCMB assertion.

If subsequent matches occur on comparators A1 and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

At OPWMT mode entry the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

In order to achieve 0% duty cycle both registers A1 and B must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the complement value of EDPOL.

In order to achieve 100% duty cycle the register B1 must be set to a value greater than maximum value of the selected time base. As a consequence, if 100% duty cycle must be implemented, the maximum counter value for the time base is 0xFFFFE for a 16-bit counter. When a match on comparator A1 occurs the output flip-flop is set at every period to the value of EDPOL bit. The transfer from register B2 to B1 is still triggered by the match at comparator A.

*Figure 397* shows the Unified Channel running in OPWMT mode with Trigger Event Generation and duty cycle update on next period update.

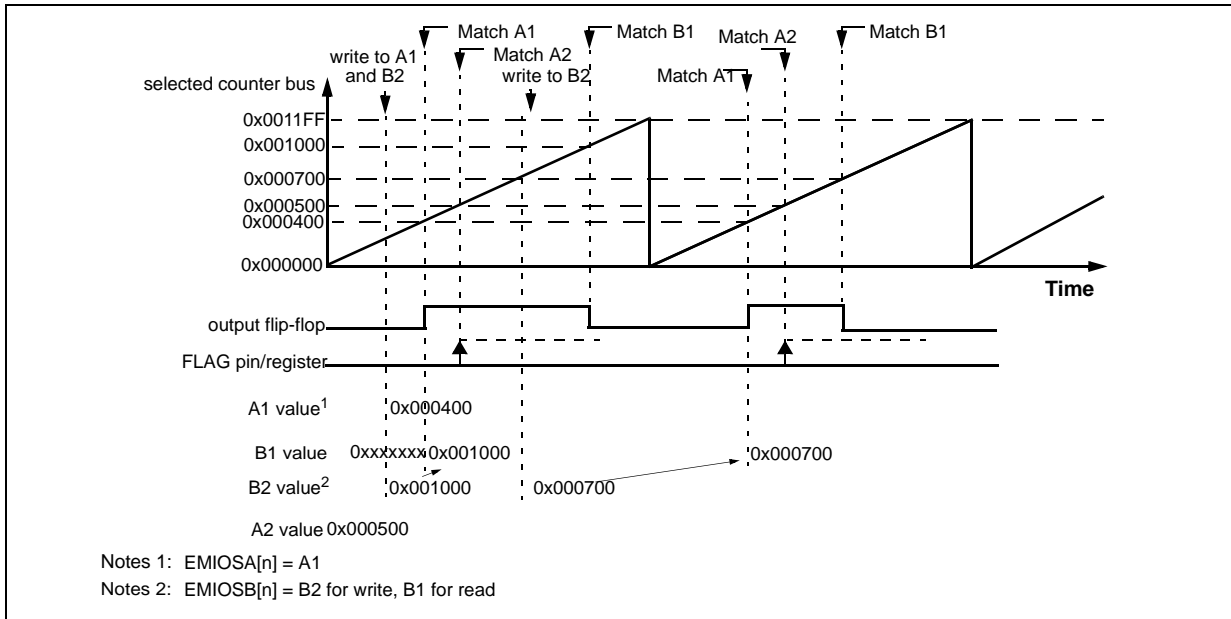


Figure 397. OPWMT example

Figure 398 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 0 % duty.

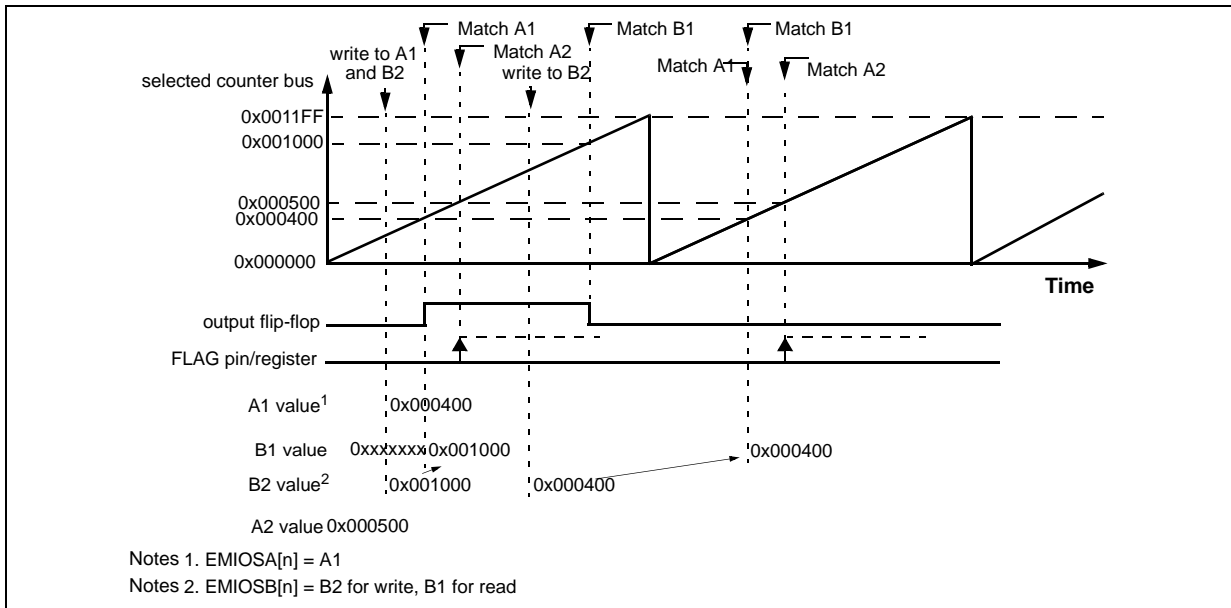


Figure 398. OPWMT with 0 % Duty Cycle

Figure 399 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 100 % duty cycle.

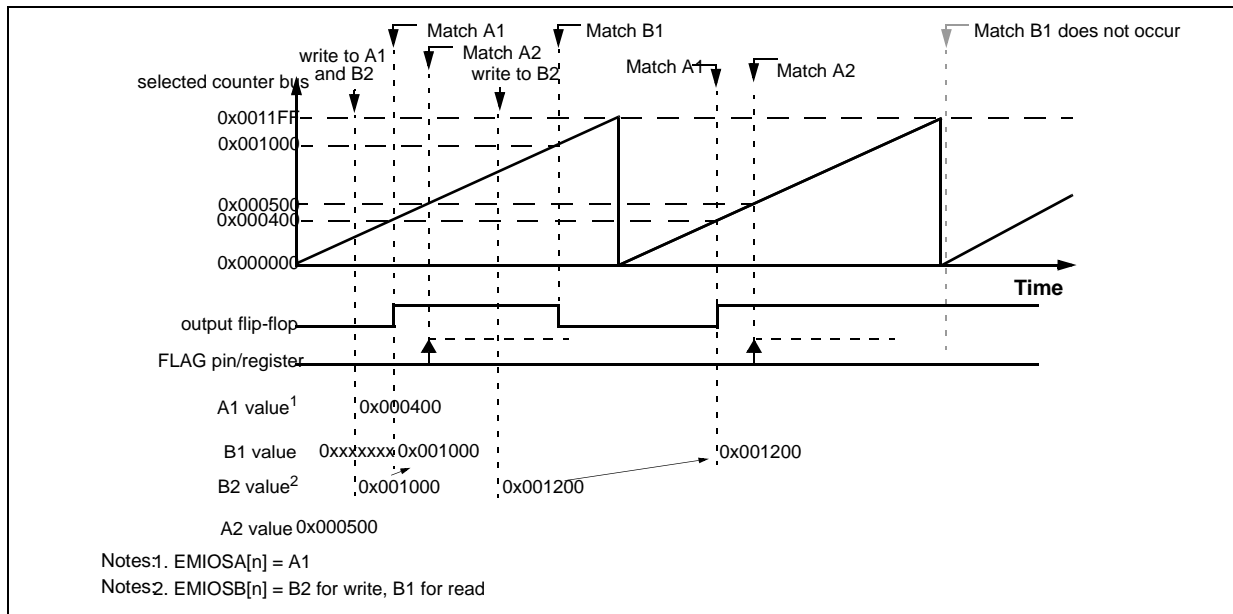


Figure 399. OPWMT with 100% duty cycle

#### 27.4.4.4.1 Input Programmable Filter (IPF)

The IPF ensures that only valid input pin transitions are received by the Unified Channel edge detector. A block diagram of the IPF is shown in [Figure 400](#).

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF[0:3] in EMIOSC[n] register.

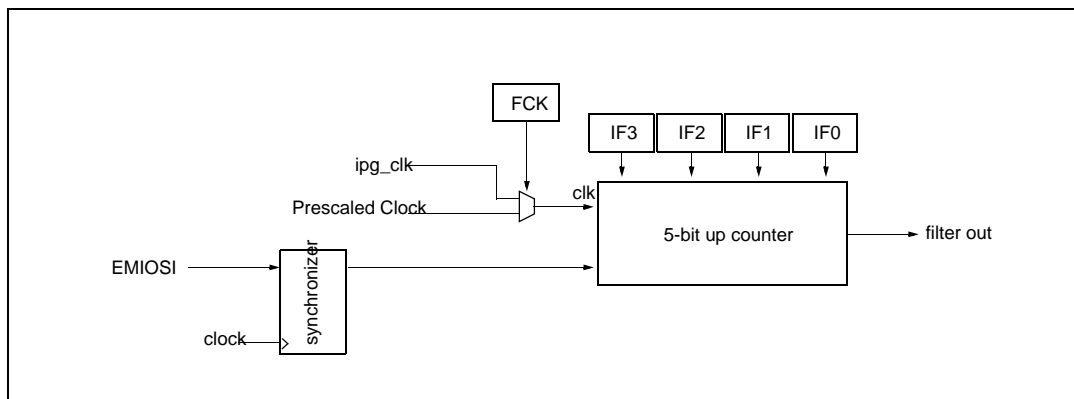
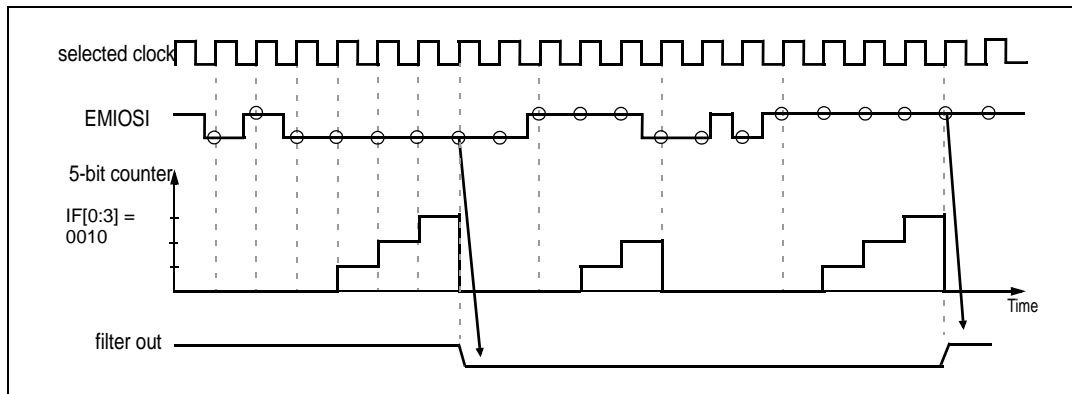


Figure 400. Input programmable filter submodule diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflow occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. A timing diagram of the input filter is shown in [Figure 401](#).



**Figure 401. Input programmable filter example**

The filter is not disabled during either freeze state or negated GTBE input.

#### 27.4.4.4.2 Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the Unified Channels. The GCP output signal is prescaled by the value defined in [Table 361](#) according to the UCPRE[0:1] bits in EMIOSC[n] register. The prescaler is enabled by setting the UCPREN bit in the EMIOSC[n] and can be stopped at any time by clearing this bit, thereby stopping the internal counter in the Unified Channel.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at both GPREN bit in EMIOSMCR register and UCPREN bit in EMIOSC[n] register, thus disabling prescalers;
2. Write the desired value for prescaling rate at UCPRE[0:1] bits in EMIOSC[n] register;
3. Enable channel prescaler by writing 1 at UCPREN bit in EMIOSC[n] register;
4. Enable global prescaler by writing 1 at GPREN bit in EMIOSMCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

#### 27.4.4.4.3 Effect of Freeze on the Unified Channel

When in debug mode, bit FRZ in the EMIOSMCR and bit FREN in the EMIOSC[n] register are both set, the internal counter and Unified Channel capture and compare functions are halted. The UC is frozen in its current state.

During freeze, all registers are accessible. When the Unified Channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

Note that for input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or freeze enable bit is cleared (FRZ in the EMIOSMCR or FREN in the EMIOSC[n] register) the channel actions resume, but may be inconsistent until channel enters GPIO mode again.

### 27.4.4.5 IP Bus Interface Unit (BIU)

The BIU provides the interface between the Internal Interface Bus (IIB) and the Peripheral Bus, allowing communication among all submodules and this IP interface.

The BIU allows 8, 16 and 32-bit access. They are performed over a 32-bit data bus in a single cycle clock.

#### 27.4.4.5.1 Effect of Freeze on the BIU

When the FRZ bit in the EMIOSMCR is set and the module is in debug mode, the operation of BIU is not affected.

### 27.4.4.6 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the channels. The main clock signal is prescaled by the value defined in [Table 355](#) according to bits GPRE[0:7] in the EMIOSMCR. The global prescaler is enabled by setting the GPREN bit in the EMIOSMCR and can be stopped at any time by clearing this bit, thereby stopping the internal counters in all the channels.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write '0' at GPREN bit in EMIOSMCR, thus disabling global prescaler;
2. Write the desired value for prescaling rate at GPRE[0:7] bits in EMIOSMCR;
3. Enable global prescaler by writing '1' at GPREN bit in EMIOSMCR.

The prescaler is not disabled during either freeze state or negated GTBE input.

#### 27.4.4.6.1 Effect of Freeze on the GCP

When the FRZ bit in the EMIOSMCR is set and the module is in debug mode, the operation of GCP submodule is not affected, that is, there is no freeze function in this submodule.

## 27.4.5 Initialization/Application information

On resetting the eMIOS the Unified Channels enter GPIO input mode.

### 27.4.5.1 Considerations

Before changing an operating mode, the UC must be programmed to GPIO mode and EMIOSA[n] and EMIOSB[n] registers must be updated with the correct values for the next operating mode. Then the EMIOSC[n] register can be written with the new operating mode. If a UC is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, that is, matches can occur in random time if the contents of EMIOSA[n] or EMIOSB[n] were not updated with the correct value before the time base matches the previous contents of EMIOSA[n] or EMIOSB[n].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

### 27.4.5.2 Application information

Correlated output signals can be generated by all output operation modes. Bits OU[n] of the EMIOSOUDIS register can be used to control the update of these output signals.

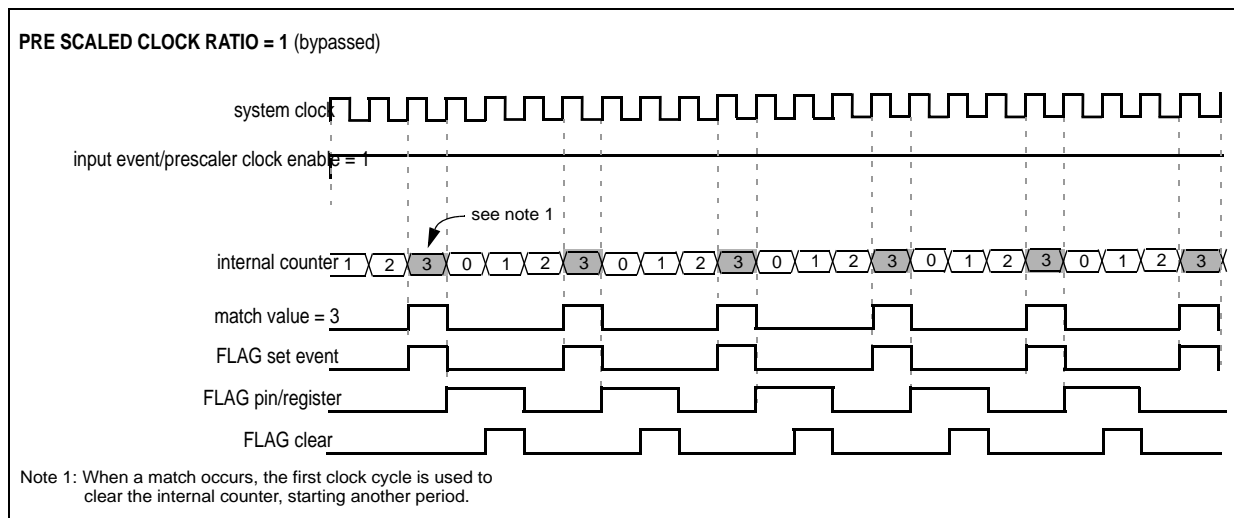


In order to guarantee that the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set up before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio, but at a different clock cycle.

**27.4.5.2.1 Time base generation**

For MC with internal clock source operation modes, the internal counter rate can be modified by configuring the clock prescaler ratio. *Figure 402* shows an example of a time base with prescaler ratio equal to one.

*Note:* *MCB and OPWFMB modes have a different behavior.*

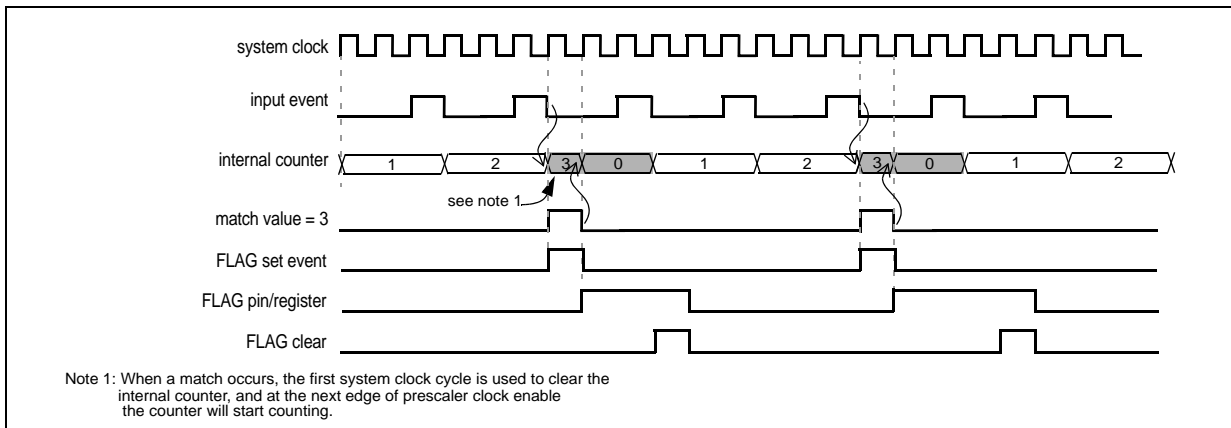


**Figure 402. Time base period when running in the fastest prescaler ratio**

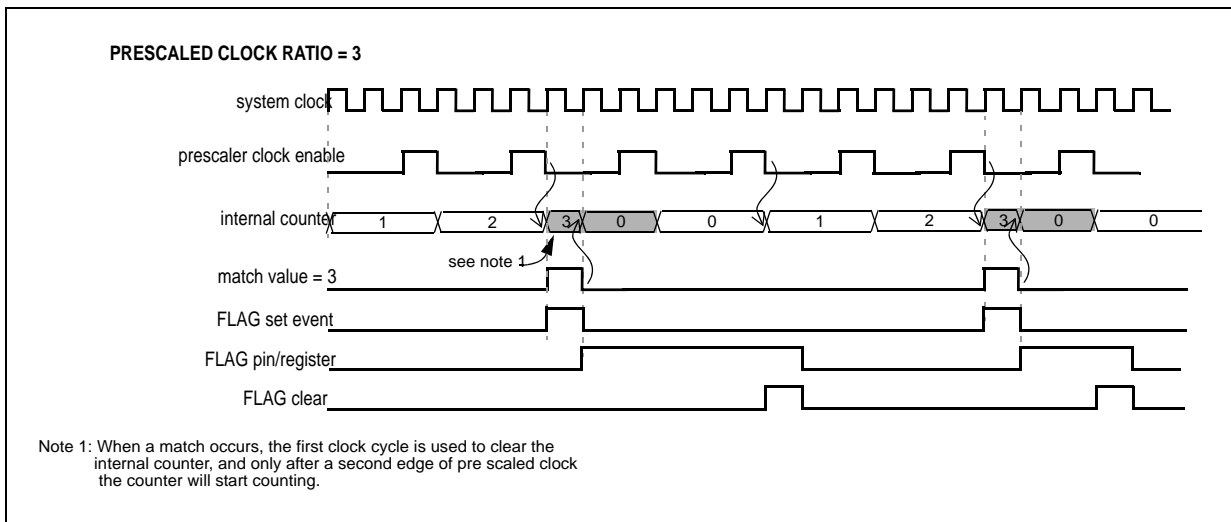
If the prescaler ratio is greater than one or external clock is selected, the counter may behave in three different ways depending on the channel mode:

- If MC mode and Clear on Match Start and External Clock source are selected the internal counter behaves as described in *Figure 403*.
- If MC mode and Clear on Match Start and Internal Clock source are selected the internal counter behaves as described in *Figure 404*.
- If MC mode and Clear on Match End are selected the internal counter behaves as described in *Figure 405*.

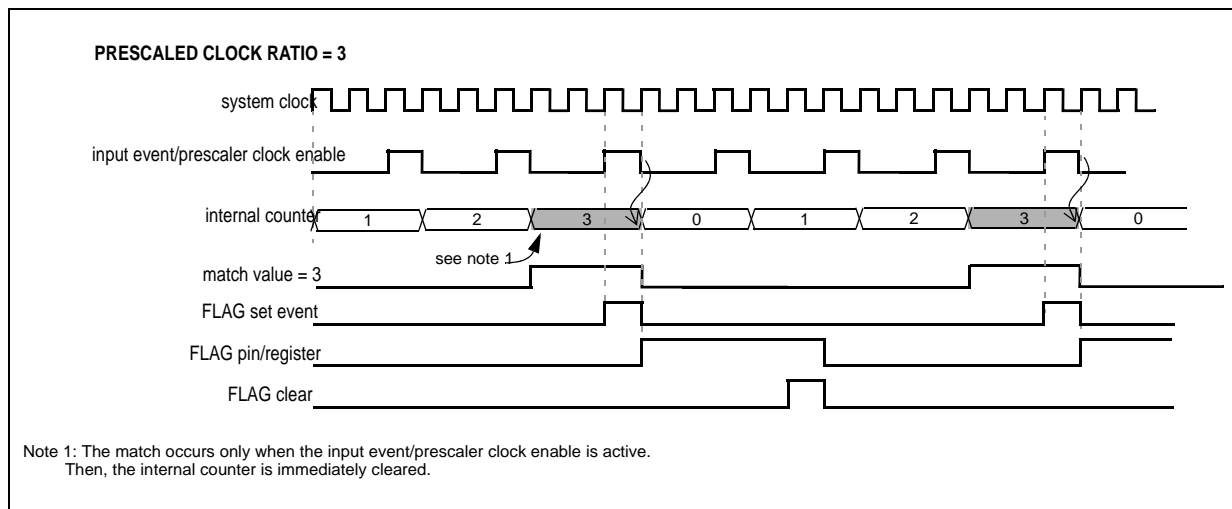
*Note:* *MCB and OPWFMB modes have a different behavior.*



**Figure 403. Time base generation with external clock and clear on match start**



**Figure 404. Time base generation with internal clock and clear on match start**



**Figure 405. Time base generation with clear on match end**

### 27.4.5.2.2 Coherent accesses

It is highly recommended that the software waits for a new FLAG set event before start reading EMIOSA[n] and EMIOSB[n] registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt request or DMA request or CTU trigger generation.

Reading the EMIOSA[n] register again in the same period of the last read of EMIOSB[n] register may lead to incoherent results. This will occur if the last read of EMIOSB[n] register occurred after a disabled B2 to B1 transfer.

### 27.4.5.2.3 Channel/Modes initialization

The following basic steps summarize basic output mode startup, assuming the channels are initially in GPIO mode:

1. [global] Disable Global Prescaler.
2. [timebase channel] Disable Channel Prescaler.
3. [timebase channel] Write initial value at internal counter.
4. [timebase channel] Set A/B register.
5. [timebase channel] Set channel to MC(B) Up mode.
6. [timebase channel] Set prescaler ratio.
7. [timebase channel] Enable Channel Prescaler.
8. [output channel] Disable Channel Prescaler.
9. [output channel] Set A/B register.
10. [output channel] Select timebase input through bits BSL[1:0].
11. [output channel] Enter output mode.
12. [output channel] Set prescaler ratio (same ratio as timebase channel).
13. [output channel] Enable Channel Prescaler.

- 14. [global] Enable Global Prescaler.
- 15. [global] Enable Global Time Base.

The timebase channel and the output channel may be the same for some applications such as in OPWFM(B) mode or whenever the output channel is intended to run the timebase itself.

The flags can be configured at any time.

## 27.5 Periodic Interrupt Timer (PIT)

### 27.5.1 Introduction

The PIT is an array of timers that can be used to raise interrupts and trigger DMA channels.

Figure 406 shows the PIT block diagram.

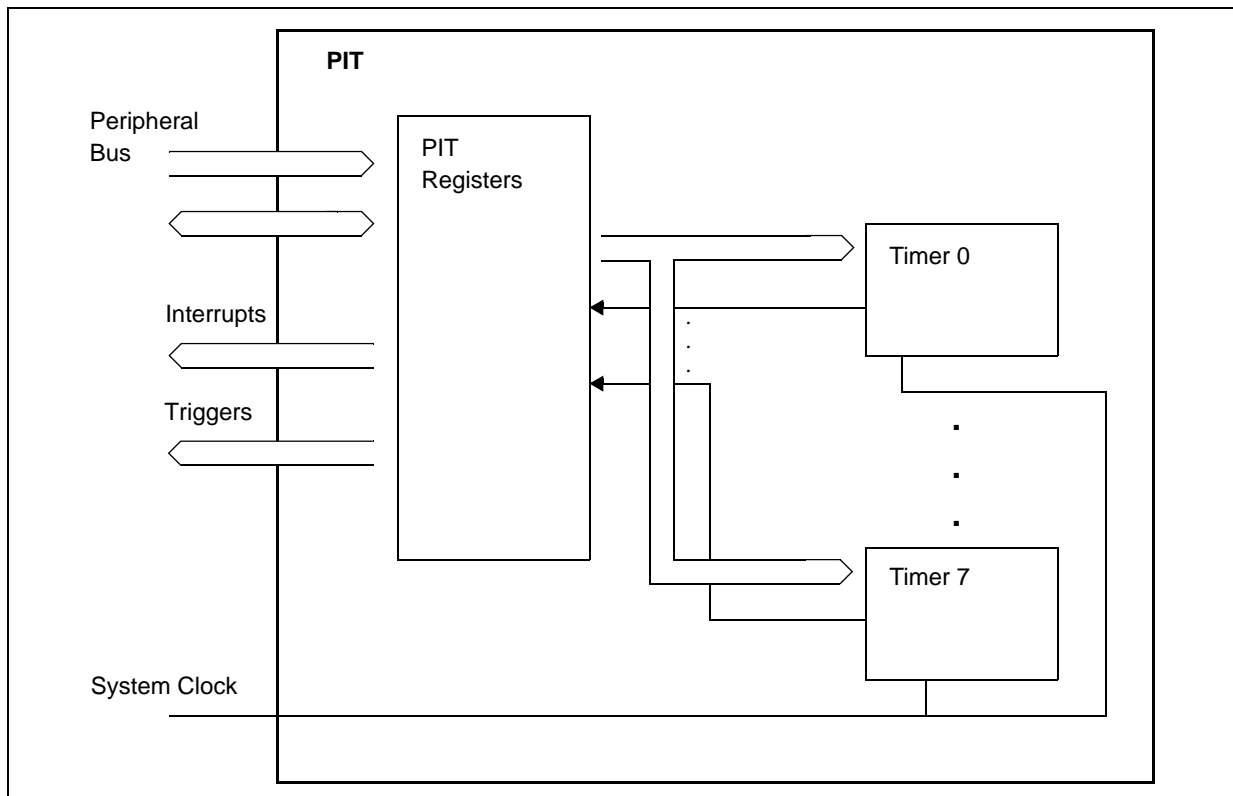


Figure 406. PIT block diagram

### 27.5.2 Features

The main features of this block are:

- Timers can generate DMA trigger pulses
- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

### 27.5.3 Signal description

The PIT module has no external pins.

### 27.5.4 Memory map and register description

This section provides a detailed description of all registers accessible in the PIT module.

#### 27.5.4.1 Memory map

[Table 366](#) gives an overview of the PIT registers. See the chip memory map for the PIT base address.

**Table 366. PIT memory map**

Base address: 0xC3FF_0000		
Address offset	Use	Location
0x000	PIT Module Control Register (PITMCR)	<a href="#">on page 750</a>
0x004–0x0FC	Reserved	
0x100–0x10C	Timer Channel 0	See <a href="#">Table 367</a>
0x110–0x11C	Timer Channel 1	See <a href="#">Table 367</a>
0x120–0x12C	Timer Channel 2	See <a href="#">Table 367</a>
0x130–0x13C	Timer Channel 3	See <a href="#">Table 367</a>
0x140–0x14C	Timer Channel 4	See <a href="#">Table 367</a>
0x150–0x15C	Timer Channel 5	See <a href="#">Table 367</a>
0x160–0x16C	Timer Channel 6	See <a href="#">Table 367</a>
0x170–0x17C	Timer Channel 7	See <a href="#">Table 367</a>

**Table 367. Timer channel *n***

Address offset	Use	Location
channel + 0x00	Timer Load Value Register (LDVAL)	<a href="#">on page 750</a>
channel + 0x04	Current Timer Value Register (CVAL)	<a href="#">on page 751</a>
channel + 0x08	Timer Control Register (TCTRL)	<a href="#">on page 752</a>
channel + 0x0C	Timer Flag Register (TFLG)	<a href="#">on page 752</a>

Note: Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.

Note: Reserved registers will read as 0, writes will have no effect.

**27.5.4.2 PIT Module Control Register (PITMCR)**

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Offset: 0x000 Access: Read/Write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

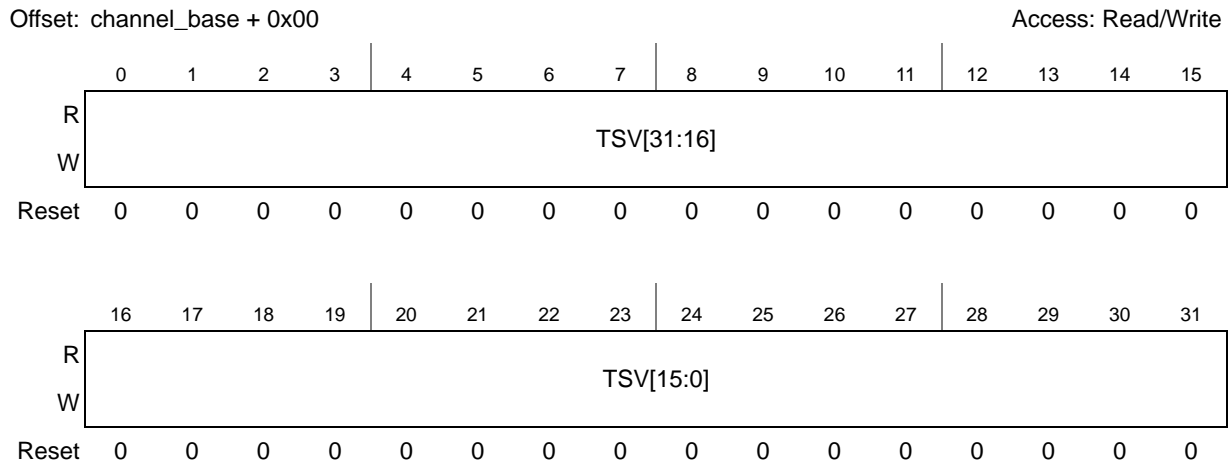
**Figure 407. PIT Module Control Register (PITMCR)**

**Table 368. PITMCR field descriptions**

Field	Description
MDIS	Module Disable This is used to disable the module clock. This bit should be enabled before any other setup is done. 0 Clock for PIT timers is enabled 1 Clock for PIT timers is disabled (default)
FRZ	Freeze Allows the timers to be stopped when the device enters debug mode. 0 = Timers continue to run in debug mode. 1 = Timers are stopped in debug mode.

**27.5.4.3 Timer Load Value Register (LDVAL)**

This register selects the timeout period for the timer interrupts.



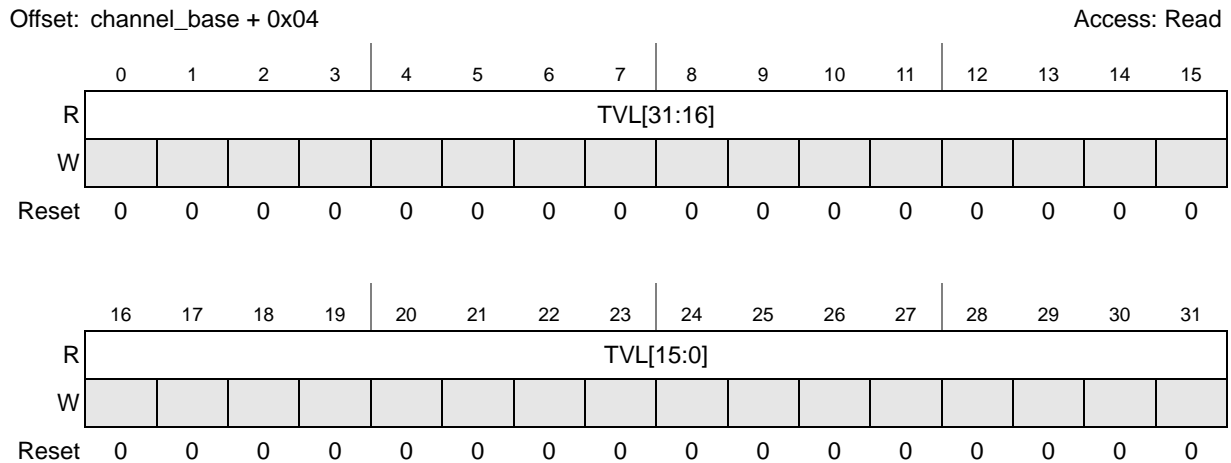
**Figure 408. Timer Load Value Register (LDVAL)**

**Table 369. LDVAL field descriptions**

Field	Description
TSV	Time Start Value This field sets the timer start value. The timer counts down until it reaches 0, then it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer, instead the value is loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see <a href="#">Figure 413</a> ).

**27.5.4.4 Current Timer Value Register (CVAL)**

This register indicates the current timer position.



**Figure 409. Current Timer Value Register (CVAL)**

**Table 370. CVAL field descriptions**

Field	Description
TVL	<p>Current Timer Value                      This field represents the current timer value. Note that the timer uses a downcounter.</p> <p><b>Note:</b> The timer values will be frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see <a href="#">Figure 352</a>).</p>

**27.5.4.5 Timer Control Register (TCTRL)**

This register contains the control bits for each timer.

Offset: channel\_base + 0x08

Access: Read/Write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 410. Timer Control Register (TCTRL)**

**Table 371. TCTRL field descriptions**

Field	Description
TIE	<p>Timer Interrupt Enable Bit                      0 Interrupt requests from Timer x are disabled                      1 Interrupt will be requested whenever TIF is set                      When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.</p>
TEN	<p>Timer Enable Bit                      0 Timer will be disabled                      1 Timer will be active</p>

**27.5.4.6 Timer Flag Register (TFLG)**

This register holds the PIT interrupt flags.



Offset: channel\_base + 0x0C

Access: Read/Write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 411. Timer Flag Register (TFLG)

Table 372. TFLG field descriptions

Field	Description
TIF	Time Interrupt Flag TIF is set to 1 at the end of the timer period. This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request. 0 Time-out has not yet occurred 1 Time-out has occurred

## 27.5.5 Functional description

### 27.5.5.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts, each interrupt will be available on a separate interrupt line.

#### 27.5.5.1.1 Timers

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their LDVAL registers, then count down until they reach 0. Then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL registers). A new interrupt can be generated only after the previous one is cleared.

If desired, the current counter value of the timer can be read via the CVAL registers.

The counter period can be restarted, by first disabling, then enabling the timer with the TEN bit (see [Figure 412](#)).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see [Figure 413](#)).

It is also possible to change the counter period without restarting the timer by writing the LDVAL register with the new load value. This value will then be loaded after the next trigger event (see [Figure 414](#)).

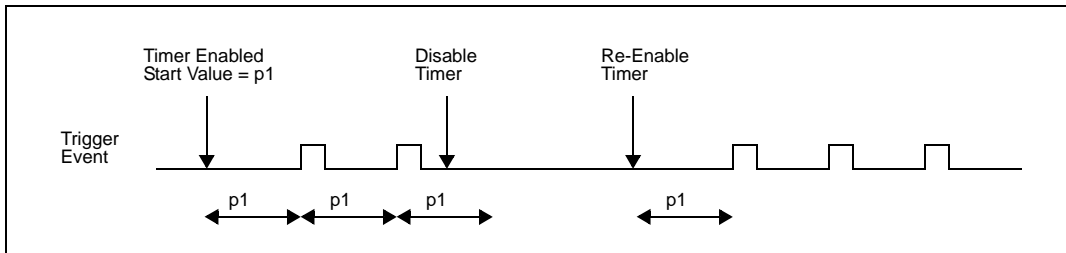


Figure 412. Stopping and starting a timer

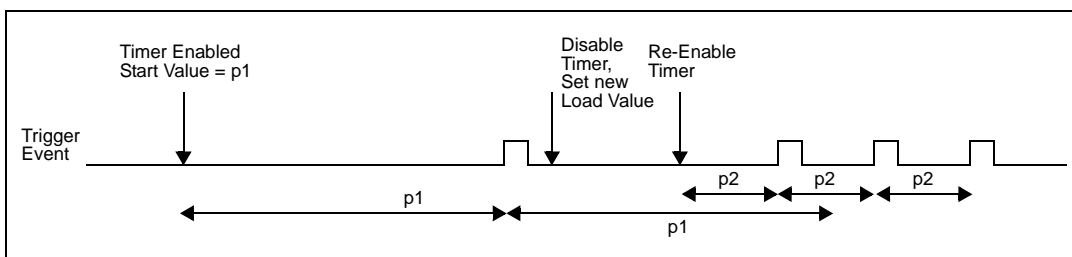


Figure 413. Modifying running timer period

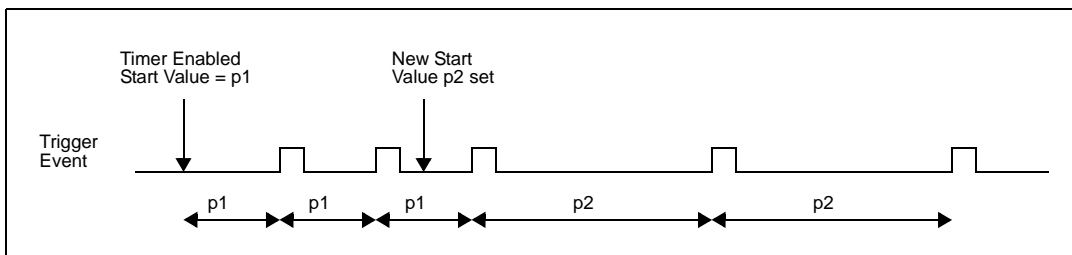


Figure 414. Dynamically setting a new load value

### 27.5.5.1.2 Debug mode

In Debug mode the timers will be frozen. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (for example, the timer values) and then continue the operation.

### 27.5.5.2 Interrupts

All of the timers support interrupt generation. See the INTC chapter of the reference manual for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

## 27.5.6 Initialization and application information

### 27.5.6.1 Example configuration

In the example configuration:

- The PIT clock has a frequency of 50 MHz
- Timer 1 creates an interrupt every 5.12 ms
- Timer 3 creates a trigger event every 30 ms

First the PIT module needs to be activated by programming `PIT_MCR[MDIS] = 0`.

The 50 MHz clock frequency equates to a clock period of 20 ns. Timer 1 needs to trigger every  $5.12 \text{ ms} / 20 \text{ ns} = 256000$  cycles and Timer 3 every  $30 \text{ ms} / 20 \text{ ns} = 1500000$  cycles. The value for the LDVAL register trigger would be calculated as  $(\text{period} / \text{clock period}) - 1$ .

The LDVAL registers must be set as follows:

- LDVAL for Timer 1 is set to `0x0003E7FF`
- LDVAL for Timer 3 is set to `0x0016E35F`

The interrupt for Timer 1 is enabled by setting TIE in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register; bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3 for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

## 28 Analog-to-Digital Converter (ADC)

### 28.1 Overview

#### 28.1.1 Device-specific features

- 2 ADC modules, (ADC\_0 with 10-bit resolution and ADC\_1 with 12-bit resolution)
- $0-V_{DD}$  common mode conversion range
- Independent reference supplies for each ADC
- 53 single-ended input channels (depending on package type), expandable to 81 channels via external multiplexing
  - Internally multiplexed channels
    - 16 precision channels shared between 10-bit and 12-bit ADCs
    - 3 standard channels shared between 10-bit and 12-bit ADCs
    - 5 dedicated standard channels on 12-bit ADC
    - Up to 29 dedicated standard channels on 10-bit ADC
  - Externally multiplexed channels
    - Internal control to support generation of external analog multiplexer selection
    - 4 internal channels optionally used to support externally multiplexed inputs, providing transparent control for additional ADC channels
    - Each of the 4 channels supports as many as 8 externally multiplexed inputs
- 3 independently configurable sample and conversion times for high precision channels, standard precision channels and externally multiplexed channels
- Dedicated result registers available for every channel. Conversion information, such as mode of operation (normal, injected or CTU), is associated to data value.
- One Shot/Scan Modes
- Chain Injection Mode
- Conversion triggering sources:
  - Software
  - CTU
  - PIT channel 2 and 6 (for injected conversion)
- Conversion triggering support — Internal conversion triggering from periodic interrupt timer (PIT) or timed I/O module (eMIOS)
- Power-down mode for analog portion of ADC
- Supports DMA transfer of results based on the end of conversion
- 6 + 3 analog watchdogs (6 on 10-bit ADC, 3 on 12-bit ADC) with interrupt capability for continuous hardware monitoring

28.1.2 Device-specific implementation

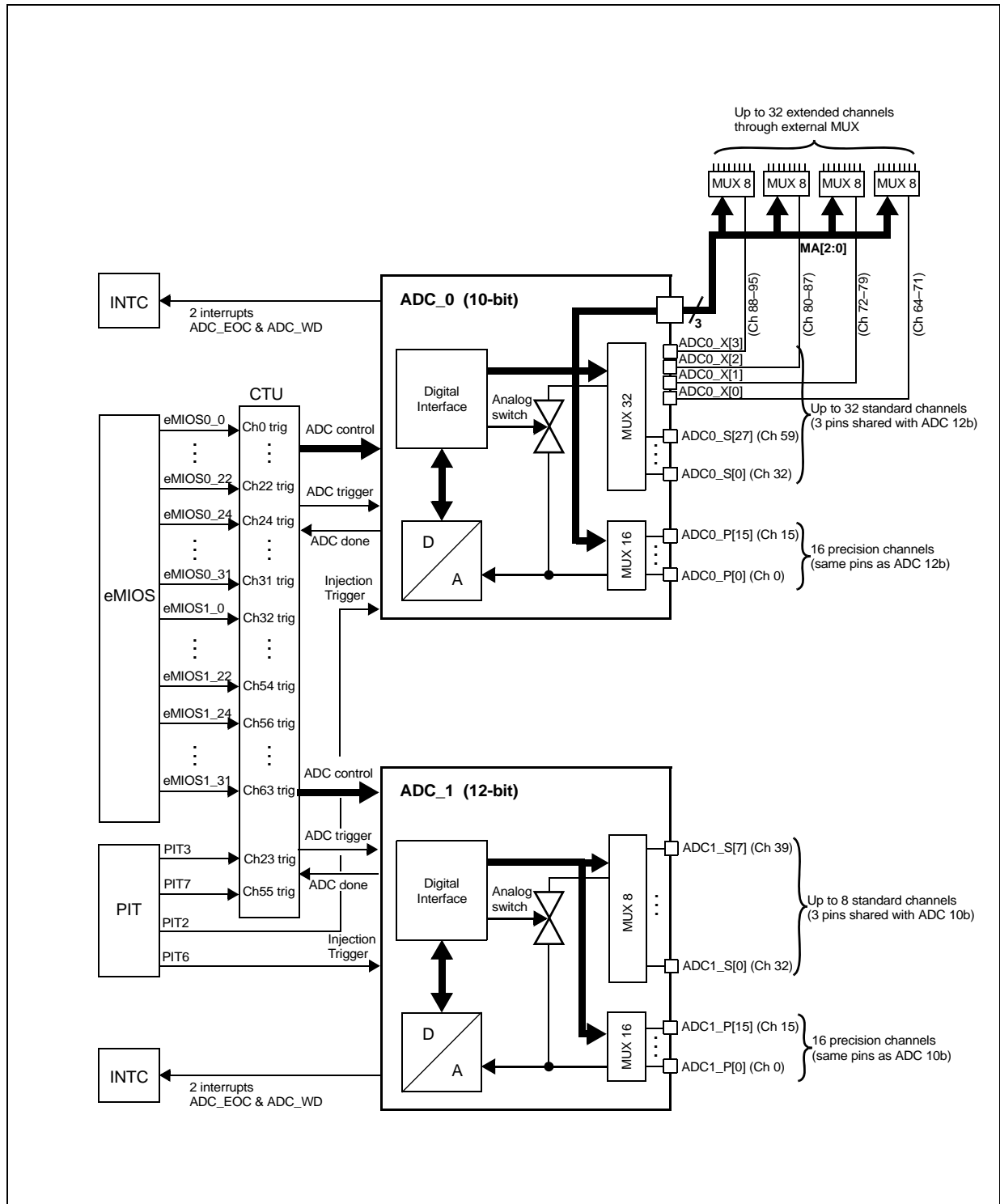


Figure 415. Implementation of ADC\_0 and ADC\_1

## 28.2 Introduction

The analog-to-digital converter (ADC) block provides accurate and fast conversions for a wide range of applications.

The ADC contains advanced features for normal or injected conversion. It provides support for eDMA (direct memory access) mode operation. A conversion can be triggered by software or hardware (Cross Triggering Unit or PIT).

There are three types of input channels:

- Internal precision, ADCx\_P[n] (internally multiplexed precision channels)
- Internal standard, ADCx\_S[n] (internally multiplexed standard channels)
- External ADCx\_X[n] (externally multiplexed standard channels)

The mask registers present within the ADC can be programmed to configure which channel has to be converted.

Three external decode signals MA[2:0] (multiplexer address) are provided for external channel selection and are available as alternate functions on GPIO.

The MA[0:2] are controlled by the ADC itself and are set automatically by the hardware.

A conversion timing register for configuring different sampling and conversion times is associated to each channel type.

Analog watchdogs allow continuous hardware monitoring.

## 28.3 Functional description

### 28.3.1 Analog channel conversion

Three conversion modes are available within the ADC:

- Normal conversion
- Injected conversion
- CTU triggered conversion

#### 28.3.1.1 Normal conversion

This is the normal conversion that the user programs by configuring the normal conversion mask registers (NCOMR). Each channel can be individually enabled by setting '1' in the corresponding field of NCOMR registers. Mask registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends (NSTART bit in the Main Status Register (MSR) is reset).

#### 28.3.1.2 Start of normal conversion

The conversion chain starts when the NSTART bit in the Main Configuration Register (MCR) is set.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

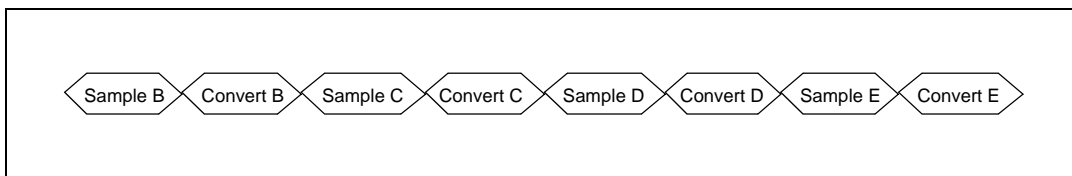
If the content of all the normal conversion mask registers is zero (that is, no channel is selected) the conversion operation is considered completed and the interrupt ECH (see interrupt controller chapter for further details) is immediately issued after the start of conversion.

**28.3.1.3 Normal conversion operating modes**

Two operating modes are available for the normal conversion:

- One Shot
- Scan

To enter one of these modes, it is necessary to program the MCR[MODE] bit. The first phase of the conversion process involves sampling the analog channel and the next phase involves the conversion phase when the sampled analog value is converted to digital as shown in *Figure 416*.



**Figure 416. Normal conversion flow**

In **One Shot Mode** (MODE = 0) a sequential conversion specified in the NCMR registers is performed only once. At the end of each conversion, the digital result of the conversion is stored in the corresponding data register.

**Example 8** One Shot Mode (MODE = 0)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the One Shot Mode. MODE = 0 is set for One Shot mode. Conversion starts from the channel B followed by conversion of channels D-E. At the end of conversion of channel E the scanning of channels stops.

The NSTART status bit in the MSR is automatically set when the Normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

In **Scan Mode** (MODE = 1), a sequential conversion of N channels specified in the NCMR registers is continuously performed. As in the previous case, at the end of each conversion the digital result of the conversion is stored into the corresponding data register.

The MSR[NSTART] status bit is automatically set when the Normal conversion starts. Unlike One Shot Mode, the MCR[NSTART] bit is not reset. It can be reset by software when the user needs to stop scan mode. In that case, the ADC completes the current scan conversion and, after the last conversion, also resets the MSR[NSTART] bit.

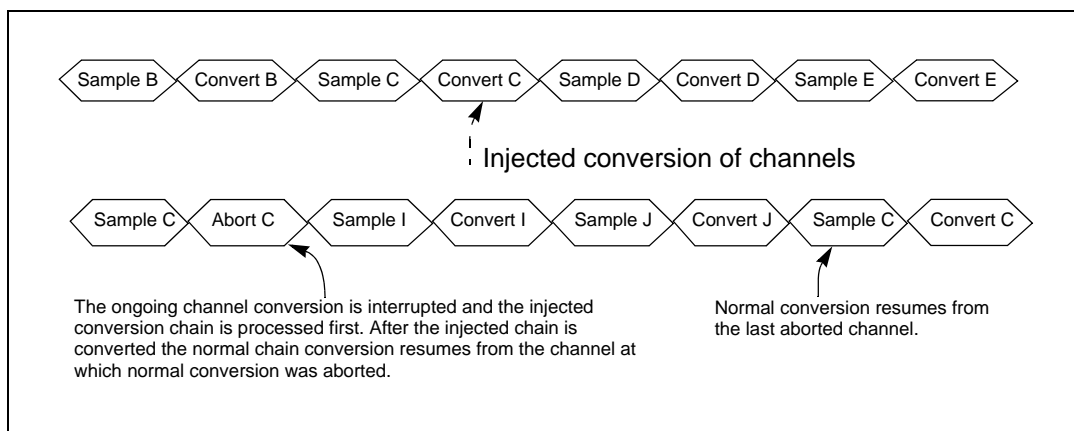
**Example 9** Scan Mode (MODE = 1)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the Scan Mode. MODE = 1 is set for Scan Mode. Conversion starts from the channel B followed by conversion of the channels D-E. At the end of conversion of channel E the scanning of channel B starts followed by conversion of the channels D-E. This sequence repeats itself till the MCR[NSTART] bit is cleared by software.

At the end of each conversion an End Of Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the conversion sequence an End Of Chain interrupt is issued (if enabled by the corresponding mask bit in the IMR register).

### 28.3.1.4 Injected channel conversion

A conversion chain can be injected into the ongoing Normal conversion by configuring the Injected Conversion Mask Registers (JCMR). As Normal conversion, each channel can be individually selected. This injected conversion (which can only occur in One Shot mode) interrupts the normal conversion (which can be in One Shot or Scan mode). When an injected conversion is inserted, ongoing normal channel conversion is aborted and the injected channel request is processed. After the last channel in the injected chain is converted, normal conversion resumes from the channel at which the normal conversion was aborted as shown in *Figure 417*.



**Figure 417. Injected sample/conversion sequence**

The injected conversion can be started using two options:

- By software setting the MCR[JSTART]; the current conversion is suspended and the injected chain is converted. At the end of the chain, the JSTART bit in the MSR is reset and the normal chain conversion is resumed.
- By an internal trigger signal from the PIT when MCR[JTRGEN] is set; a programmed event (rising/falling edge depending on MCR[JEDGE]) on the signal coming from PIT starts the injected conversion by setting the MSR[JSTART]. At the end of the chain, the MSR[JSTART] is cleared and the normal conversion chain is resumed.

The MSR[JSTART] is automatically set when the Injected conversion starts. At the same time the MCR[JSTART] is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running injected conversion is completed.

At the end of each injected conversion, an End Of Injected Conversion (JEOC) interrupt is issued (if enabled by the IMR[MSKJEOC]) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the IMR[MSKJEOC]).

If the content of all the injected conversion mask registers (JCMR) is zero (that is, no channel is selected) the JECH interrupt is immediately issued after the start of conversion.



### 28.3.1.5 Abort conversion

Two different abort functions are provided.

- The user can abort the ongoing conversion by setting the MCR[ABORT] bit. The current conversion is aborted and the conversion of the next channel of the chain is immediately started. In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit is reset after the conversion of the next channel starts. The EOC interrupt corresponding to the aborted channel is not generated. This behavior is true for normal or Injected conversion modes. If the last channel of a chain is aborted, the end of chain is reported generating an ECH interrupt.
- It is also possible to abort the current chain conversion by setting the MCR[ABORTCHAIN] bit. In that case the behavior of the ADC depends on the MODE bit. If scan mode is disabled, the NSTART bit is automatically reset together with the MCR[ABORTCHAIN] bit. Otherwise, if the scan mode is enabled, a new chain conversion is started. The EOC interrupt of the current aborted conversion is not generated but an ECH interrupt is generated to signal the end of the chain.

When a chain conversion abort is requested (ABORTCHAIN bit is set) while an injected conversion is running over a suspended Normal conversion, both injected chain and Normal conversion chain are aborted (both the NSTART and JSTART bits are also reset).

### 28.3.2 Analog clock generator and conversion timings

The clock frequency can be selected by programming the MCR[ADCLKSEL]. When this bit is set to '1' the ADC clock has the same frequency as the peripheral set 3 clock. Otherwise, the ADC clock is half of the peripheral set 3 clock frequency. The ADCLKSEL bit can be written only in power-down mode.

When the internal divider is not enabled (ADCCLKSEL = 1), it is important that the associated clock divider in the clock generation module is '1'. This is needed to ensure 50% clock duty cycle.

The direct clock should basically be used only in low power mode when the device is using only the 16 MHz fast internal RC oscillator, but the conversion still requires a 16 MHz clock (an 8 MHz clock is not fast enough).

In all other cases, the ADC should use the clock divided by two internally.

### 28.3.3 ADC sampling and conversion timing

In order to support different loading and switching times, several different Conversion Timing registers (CTR) are present. There is one register per channel type. INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.

When a conversion is started, the ADC connects the internal sampling capacitor to the respective analog input pin, allowing the capacitance to charge up to the input voltage value. The time to load the capacitor is referred to as sampling time. After completion of the sampling phase, the evaluation phase starts and all the bits corresponding to the resolution of the ADC are estimated to provide the conversion result.

The conversion times are programmed via the bit fields of the CTR. Bit fields INPLATCH, INPCMP and INPSAMP are used to define the total conversion duration ( $T_{conv}$ ) and in particular the partition between sampling phase duration ( $T_{sample}$ ) and total evaluation phase duration ( $T_{eval}$ ).

28.3.3.1 ADC\_0

Figure 418 represents the sampling and conversion sequence.

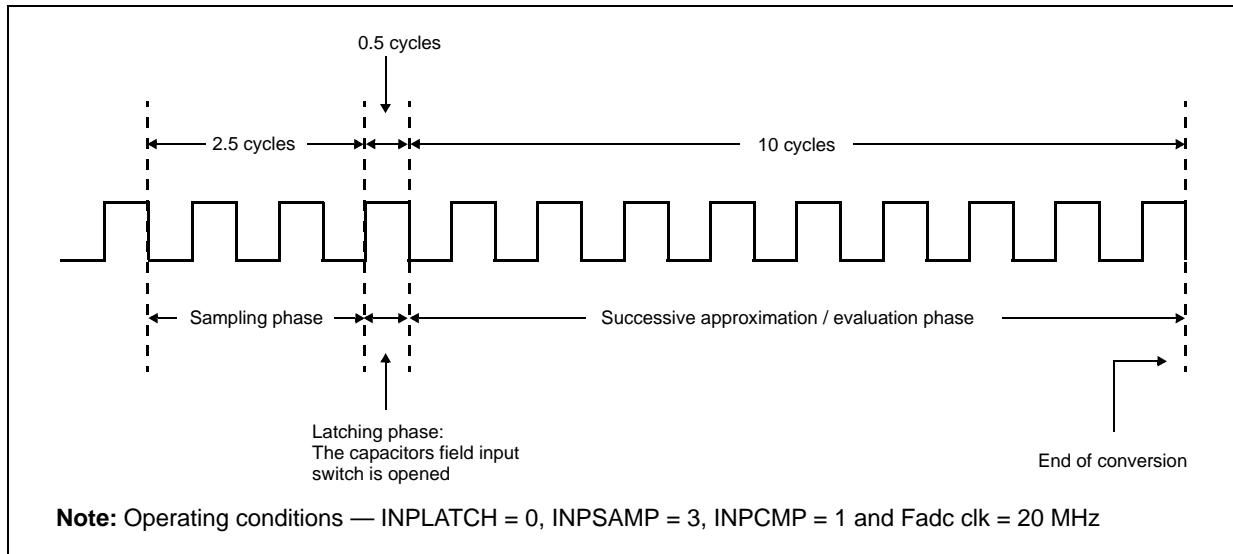


Figure 418. Sampling and conversion timings

The sampling phase duration is:

$$T_{\text{sample}} = (\text{INPSAMP} - \text{ndelay}) \cdot T_{\text{ck}}$$

$$\text{INPSAMP} \geq 3$$

where ndelay is equal to 0.5 if INPSAMP is less than or equal to 06h, otherwise it is 1. INPSAMP must be greater than or equal to 3 (hardware requirement).

The total evaluation phase duration is:

$$T_{\text{eval}} = 10 \cdot T_{\text{biteval}} = 10 \cdot (\text{INPCMP} \cdot T_{\text{ck}})$$

$$(\text{INPCMP} \geq 1) \quad \text{and} \quad (\text{INPLATCH} < \text{INPCMP})$$

INPCMP must be greater than or equal to 1 and INPLATCH must be less than INPCMP (hardware requirements).

The total conversion duration is (not including external multiplexing):

$$T_{\text{conv}} = T_{\text{sample}} + T_{\text{eval}} + (\text{ndelay} \cdot T_{\text{ck}})$$

The timings refer to the unit  $T_{\text{ck}}$ , where  $f_{\text{ck}} = (1/2 \times \text{ADC peripheral set clock})$ .

Table 373. ADC sampling and conversion timing at 5 V / 3.3 V for ADC\_0

Clock (MHz)	T <sub>ck</sub> (μs)	INPSAMPLE <sup>(1)</sup>	Ndelay <sup>(2)</sup>	T <sub>sample</sub> <sup>(3)</sup>	T <sub>sample</sub> /T <sub>ck</sub>	INPCMP	T <sub>eval</sub> (μs)	INPLATCH	T <sub>conv</sub> (μs)	T <sub>conv</sub> /T <sub>ck</sub>
6	0.167	4	0.5	0.583	3.500	1	1.667	0	2.333	14.000
7	0.143	4	0.5	0.500	3.500	1	1.429	0	2.000	14.000
8	0.125	5	0.5	0.563	4.500	1	1.250	0	1.875	15.000
16	0.063	9	1	0.500	8.000	1	0.625	0	1.188	19.000
32	0.031	17	1	0.500	16.000	2	0.625	1	1.156	37.000

- Where: INPSAMPLE ≥ 3
- Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1
- Where: T<sub>sample</sub> = (INPSAMP-N)T<sub>ck</sub>; Must be ≥ 500 ns

Table 374. Max/Min ADC\_clk frequency and related configuration settings at 5 V / 3.3 V for ADC\_0

INPCMP	INPLATCH	Max f <sub>ADC_clk</sub>	Min f <sub>ADC_clk</sub>
00/01	0	20+4%	6
	1	—	—
10	0	—	—
	1	32+4%	6
11	0	—	—
	1	32+4%	9

28.3.3.2 ADC\_1

Figure 419 represents the sampling and conversion sequence.

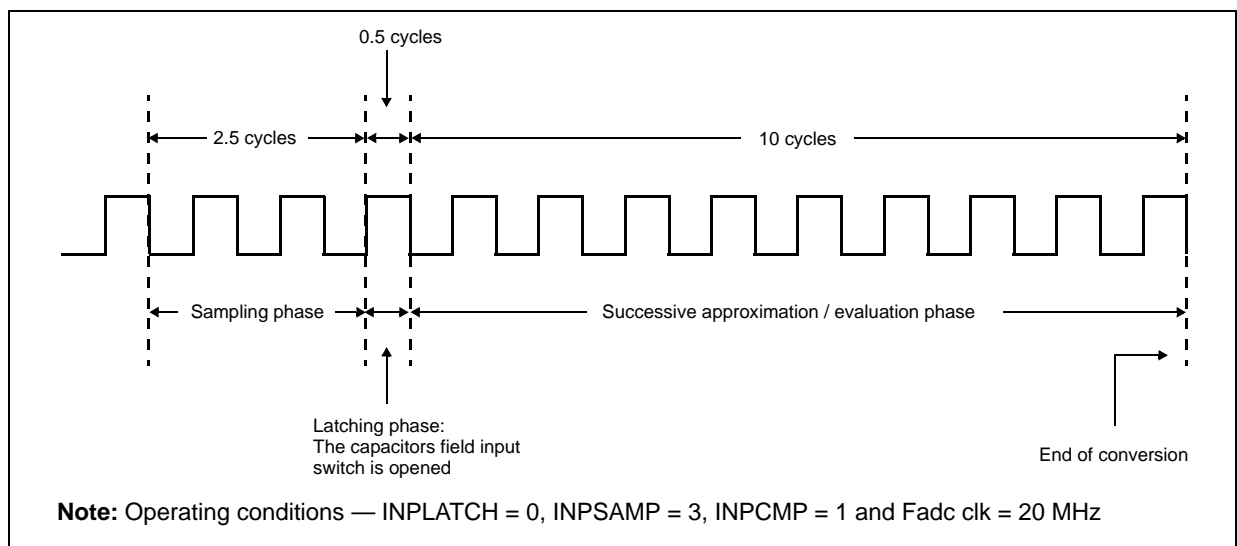


Figure 419. Sampling and conversion timings

The sampling phase duration is:

$$T_{\text{sample}} = (\text{INPSAMP} - 1) \cdot T_{\text{ck}}$$

$$\text{INPSAMP} \geq 8$$

where ndelay is equal to 0.5 if INPSAMP is less than or equal to 06h, otherwise it is 1. INPSAMP must be greater than or equal to 8 (hardware requirement).

The total evaluation phase duration is:

$$T_{\text{eval}} = 12 \cdot T_{\text{biteval}}$$

Where:

$$T_{\text{biteval}} = \text{INPCMP} \cdot T_{\text{ck}} \quad [\text{if } \text{INPCMP} \geq 1]$$

$$T_{\text{biteval}} = 4 \cdot T_{\text{ck}} \quad [\text{if } \text{INPCMP} = 0]$$

The total conversion duration is (not including external multiplexing):

$$T_{\text{conv}} = T_{\text{sample}} + T_{\text{eval}} + T_{\text{ck}}$$

The timings refer to the unit  $T_{\text{ck}}$ , where  $f_{\text{ck}} = (1/2 \times \text{ADC peripheral set clock})$ .

**Table 375. ADC sampling and conversion timing at 5 V for ADC\_1**

Clock (MHz)	T <sub>ck</sub> (μs)	INPSAMPLE <sup>(1)</sup>	Ndelay <sup>(2)</sup>	T <sub>sample</sub> <sup>(3)</sup>	T <sub>sample</sub> /T <sub>ck</sub>	INPCMP	T <sub>eval</sub> (μs)	INPLATCH	T <sub>conv</sub> (μs)	T <sub>conv</sub> /T <sub>ck</sub>
4	0.250	8	1	1.750	7.000	1	3.000	1	5.000	20.000
5	0.200	8	1	1.400	7.000	1	2.400	1	4.000	20.000
6	0.167	8	1	1.167	7.000	1	2.000	1	3.333	20.000
7	0.143	8	1	1.000	7.000	1	1.714	1	2.857	20.000
8	0.125	8	1	0.875	7.000	1	1.500	1	2.500	20.000
16	0.063	9	1	0.500	8.000	2	1.500	1	2.063	33.000
32	0.031	17	1	0.500	16.000	0	1.500	1	2.031	65.000

- Where: INPSAMPLE ≥ 8
- Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1
- Where: T<sub>sample</sub> = (INPSAMP - N)T<sub>ck</sub>; Must be ≥ 500 ns

**Table 376. ADC sampling and conversion timing at 3.3 V for ADC\_1**

Clock (MHz)	T <sub>ck</sub> (μs)	INPSAMPLE <sup>(1)</sup>	Ndelay <sup>(2)</sup>	T <sub>sample</sub> <sup>(3)</sup>	T <sub>sample</sub> /T <sub>ck</sub>	INPCMP	T <sub>eval</sub> (μs)	INPLATCH	T <sub>conv</sub> (μs)	T <sub>conv</sub> /T <sub>ck</sub>
4	0.250	8	1	1.750	7.000	1	3.000	1	5.000	20.000
5	0.200	8	1	1.400	7.000	1	2.400	1	4.000	20.000
7	0.143	8	1	1.000	7.000	2	3.429	1	4.571	32.000

**Table 376. ADC sampling and conversion timing at 3.3 V for ADC\_1 (continued)**

Clock (MHz)	T <sub>ck</sub> (μs)	INPSAMPLE <sup>(1)</sup>	Ndelay <sup>(2)</sup>	T <sub>sample</sub> <sup>(3)</sup>	T <sub>sample</sub> /T <sub>ck</sub>	INPCMP	T <sub>eval</sub> (μs)	INPLATCH	T <sub>conv</sub> (μs)	T <sub>conv</sub> /T <sub>ck</sub>
8	0.125	8	1	0.875	7.000	2	3.000	1	4.000	32.000
16	0.063	11	1	0.625	10.000	0	3.000	1	3.688	59.000
20	0.050	13	1	0.600	12.000	0	2.400	1	3.050	61.000

1. Where: INPSAMPLE ≥ 8
2. Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1
3. Where: T<sub>sample</sub> = (INPSAMP-N)T<sub>ck</sub>; Must be ≥ 600 ns

**Table 377. Max/Min ADC\_clk frequency and related configuration settings at 5 V for ADC\_1**

INPCMP	INPLATCH	Max f <sub>ADC_clk</sub>	Min f <sub>ADC_clk</sub>
00	0	16+4%	13.33
	1	32+4%	13.33
01	0/1	8+4%	3.33
10	0	8+4%	6.67
	1	16+4%	6.67
11	0	16+4%	10
	1	24+4%	10

**Table 378. Max/Min ADC\_clk frequency and related configuration settings at 3.3 V for ADC\_1**

INPCMP	INPLATCH	Max f <sub>ADC_clk</sub>	Min f <sub>ADC_clk</sub>
00	0	Not allowed	Not allowed
	1	20+4%	13.33
01	0/1	5+4%	3.33
10	0	Not allowed	Not allowed
	1	10+4%	6.67
11	0	10+4%	10
	1	15+4%	10

### 28.3.4 ADC CTU (Cross Triggering Unit)

#### 28.3.4.1 Overview

The ADC cross triggering unit (CTU) is added to enhance the injected conversion capability of the ADC. The CTU is triggered by multiple input events (eMIOS and PIT) and can be used to select the channels to be converted from the appropriate event configuration register. A single channel is converted for each request.

The CTU can be enabled by setting MCR[CTUEN].

The CTU and the ADC are synchronous with the peripheral set 3 clock in both cases.

### 28.3.4.2 CTU in trigger mode

In CTU trigger mode, normal and injected conversions triggered by the CPU are still enabled.

Once the CTU event configuration register (CTU\_EVTCFGRx) is configured and the corresponding trigger from the eMIOS or PIT is received, the conversion starts. The MSR[CTUSTART] is set automatically at this point and it is also automatically reset when the CTU triggered conversion is completed.

If an injected conversion (programmed by the user by setting the JSTART bit) is ongoing and CTU conversion is triggered, then the injected channel conversion chain is aborted and only the CTU triggered conversion proceeds. By aborting the injected conversion, the MSR[JSTART] is reset. That abort is signalled through the status bit MSR[JABORT].

If a normal conversion is ongoing and a CTU conversion is triggered, then any ongoing channel conversion is aborted and the CTU triggered conversion is processed. When it is finished, the normal conversion resumes from the channel at which the normal conversion was aborted.

If another CTU conversion is triggered before the end of the conversion, that request is discarded. However, if the CTU has triggered a conversion that is still ongoing on a channel, it will buffer a second request for the channel and wait for the end of the first conversion before requesting another conversion. Thus, two conversion requests close together will both be serviced.

When a normal conversion is requested during CTU conversion (CTUSTART bit = '1'), the normal conversion starts when CTU conversion is completed (CTUSTART = '0'). Otherwise, when an Injected conversion is requested during CTU conversion, the injected conversion is discarded and the MCR[JSTART] is immediately reset.

## 28.3.5 Presampling

### 28.3.5.1 Introduction

Presampling is used to precharge or discharge the ADC internal capacitor before it starts sampling of the analog input coming from the input pins. This is useful for resetting information regarding the last converted data or to have more accurate control of conversion speed. During presampling, the ADC samples the internally generated voltage.

Presampling can be enabled/disabled on a channel basis by setting the corresponding bits in the PSR registers.

After enabling the presampling for a channel, the normal sequence of operation will be Presampling + Sampling + Conversion for that channel. Sampling of the channel can be bypassed by setting the PRECONV bit in the PSCR. When sampling of a channel is bypassed, the sampled data of internal voltage in the presampling state is converted ([Figure 420](#), [Figure 421](#)).

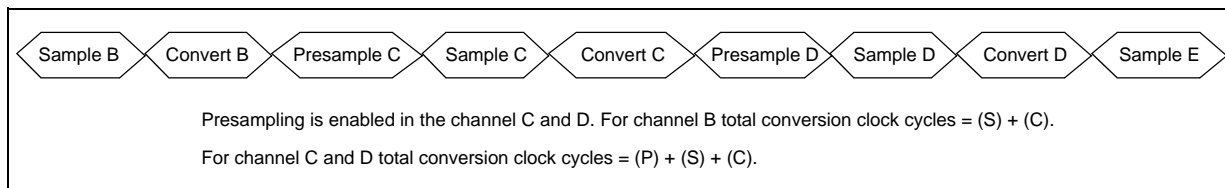


Figure 420. Presampling sequence

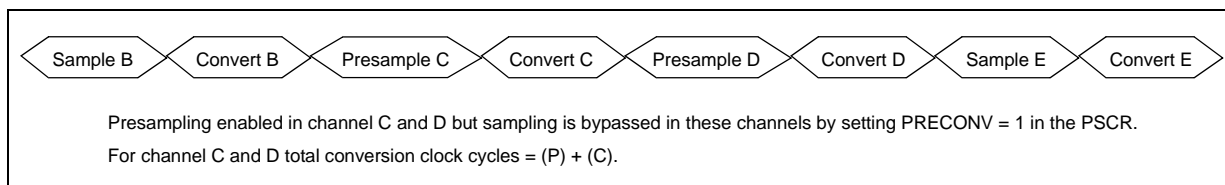


Figure 421. Presampling sequence with PRECONV = 1

### 28.3.5.2 Presampling channel enable signals

It is possible to select between two internally generated voltages V0 and V1 depending on the value of the PSCR[PREVAL] as shown in [Table 379](#).

Table 379. Presampling voltage selection based on PREVALx fields

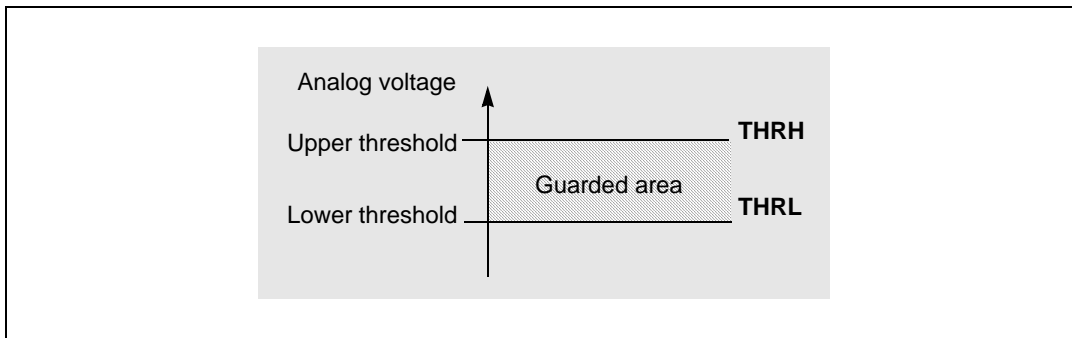
PSCR[PREVALx]	Presampling voltage
00	V0 = V <sub>SS_HV_ADC0</sub> or V <sub>SS_HV_ADC1</sub>
01	V1 = V <sub>DD_HV_ADC0</sub> or V <sub>DD_HV_ADC1</sub>
10	Reserved
11	Reserved

Three presampling value fields, one per channel type, in the PSCR make it possible to select different presampling values for each type.

## 28.3.6 Programmable analog watchdog

### 28.3.6.1 Introduction

The analog watchdogs are used for determining whether the result of a channel conversion lies within a given guarded area (as shown in [Figure 422](#)) specified by an upper and a lower threshold value named THRH and THRL respectively.



**Figure 422. Guarded area**

After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value lies outside that guarded area then corresponding threshold violation interrupts are generated. The comparison result is stored as WTISR[WDGxH] and WTISR[WDGxL] as explained in [Table 380](#). Depending on the mask bits WTIMR[MSKWDGxL] and WTIMR[MSKWDGxH], an interrupt is generated on threshold violation.

**Table 380. Values of WDGxH and WDGxL fields**

WDGxH	WDGxL	Converted data
1	0	converted data > THRH
0	1	converted data < THRL
0	0	THRL <= converted data <= THRH

Each channel can be enabled independently from the CWENR registers and can select the watchdog threshold registers (THRHLRx) to be used by programming the CWSELR registers. The threshold registers selected by the CWSELR[WSEL\_CHx] provides the threshold values.

For example, if channel number 15 is to be monitored with the threshold values in THRHLR1, then CWSELR[WSEL\_CH15] is programmed to select THRHLR1 to provide the threshold values. The channel monitoring is enabled by setting the bit corresponding to channel 15 in the CWENR.

If a converted value for a particular channel lies outside the range specified by threshold values, then the corresponding bit is set in the Analog Watchdog Out of Range Register (AWORR).

A set of threshold registers (THRHLRx) can be linked to several ADC channels. The threshold values to be selected for a channel need be programmed only once in the CWSELRx.

*Note: If the higher threshold for the analog watchdog is programmed lower than the lower threshold and the converted value is less than the lower threshold, then the WDGxL interrupt for the low threshold violation is set, else if the converted value is greater than the lower threshold (consequently also greater than the higher threshold) then the interrupt WDGxH for high threshold violation is set. Thus, the user should avoid that situation as it could lead to misinterpretation of the watchdog interrupts.*



### 28.3.7 DMA functionality

A DMA request can be programmed after the conversion of every channel by setting the respective masking bit in the DMAR registers. The DMAR masking registers must be programmed before starting any conversion. There is one DMAR per channel type and each ADC module has one DMA request associated with it.

The DMA transfers can be enabled using the DMAEN bit of DMAE register. When the DCLR bit of DMAE register is set then the DMA request is cleared on the reading of the register for which DMA transfer has been enabled.

### 28.3.8 Interrupts

The ADC generates the following two maskable interrupt signals:

- ADC\_EOC interrupt requests
  - EOC (end of conversion)
  - ECH (end of chain)
  - JEOC (end of injected conversion)
  - JECH (end of injected chain)
  - EOCTU (end of CTU conversion)
- WDGxL and WDGxH (watchdog threshold) interrupt requests

Interrupts are generated during the conversion process to signal events such as End Of Conversion as explained in register description for CEOCFR[0...2]. Two registers named CEOCFR[0...2] (Channel Pending Registers) and IMR (Interrupt Mask Register) are provided in order to check and enable the interrupt request to INT module.

Interrupts can be individually enabled on a channel by channel basis by programming the CIMR (Channel Interrupt Mask Register).

Several CEOCFR[0...2] are also provided in order to signal which of the channels' measurement has been completed.

The analog watchdog interrupts are handled by two registers WTISR (Watchdog Threshold Interrupt Status Register) and WTIMR (Watchdog Threshold Interrupt Mask Register) in order to check and enable the interrupt request to the INTC module. The Watchdog interrupt source sets two pending bits WDGxH and WDGxL in the WTISR for each of the channels being monitored.

The CEOCFR[0...2] contains the interrupt pending request status. If the user wants to clear a particular interrupt event status, then writing a '1' to the corresponding status bit clears the pending interrupt flag (at this write operation all the other bits of the CEOCFR[0...2] must be maintained at '0').

### 28.3.9 External decode signals delay

The ADC provides several external decode signals to select which external channel has to be converted. In order to take into account the control switching time of the external analog multiplexer, a Decode Signals Delay register (DSDR) is provided. The delay between the decoding signal selection and the actual start of conversion can be programmed by writing the field DSD[0:11].

After having selected the channel to be converted, the MA[0:2] control lines are automatically reset. For instance, in the event of normal scan conversion on ANP[0] followed by ANX[0,7] (ADC ch 71) all the MA[0:2] bits are set and subsequently reset.

### 28.3.10 Power-down mode

The analog part of the ADC can be put in low power mode by setting the MCR[PWDN]. After releasing the reset signal the ADC analog module is kept in power-down mode by default, so this state must be exited before starting any operation by resetting the appropriate bit in the MCR.

The power-down mode can be requested at any time by setting the MCR[PWDN]. If a conversion is ongoing, the ADC must complete the conversion before entering the power down mode. In fact, the ADC enters power-down mode only after completing the ongoing conversion. Otherwise, the ongoing operation should be aborted manually by resetting the NSTART bit and using the ABORTCHAIN bit.

MSR[ADCSTATUS] bit is set only when ADC enters power-down mode.

After the power-down phase is completed the process ongoing before the power-down phase must be restarted manually by setting the appropriate MCR[START] bit.

Resetting MCR[PWDN] bit and setting MCR[NSTART] or MCR[JSTART] bit during the same cycle is forbidden.

If a CTU trigger pulse is received during power-down, it is discarded.

If the CTU is enabled and the CSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set.

When CTU trigger mode is enabled, the application has to wait for the end of conversion (CTUSTART bit automatically reset).

### 28.3.11 Auto-clock-off mode

To reduce power consumption during the IDLE mode of operation (without going into power-down mode), an "auto-clock-off" feature can be enabled by setting the MCR[ACKO] bit. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no conversion is programmed by the user.

*Note:* The auto-clock-off feature cannot operate when the digital interface runs at the same rate as the analog interface. This means that when  $MCR.ADCCLKSEL = 1$ , the analog clock will not shut down in IDLE mode.

## 28.4 Register descriptions

### 28.4.1 Introduction

SPC560B54/6x has two ADCs (10-bit ADC\_0 and 12-bit ADC\_1) and each has specific registers.

[Table 381](#) lists the ADC\_0 registers with their address offsets and reset values.

Table 381. 10-bit ADC\_0 digital registers

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x0000	Main Configuration Register (MCR)	<a href="#">on page 777</a>
0x0004	Main Status Register (MSR)	<a href="#">on page 779</a>
0x0008...0x000F	Reserved	—
0x0010	Interrupt Status Register (ISR)	<a href="#">on page 781</a>
0x0014	Channel Pending Register (CEOCFR0)	<a href="#">on page 781</a>
0x0018	Channel Pending Register (CEOCFR1)	<a href="#">on page 781</a>
0x001C	Channel Pending Register (CEOCFR2)	<a href="#">on page 781</a>
0x0020	Interrupt Mask Register (IMR)	<a href="#">on page 783</a>
0x0024	Channel Interrupt Mask Register (CIMR0)	<a href="#">on page 784</a>
0x0028	Channel Interrupt Mask Register (CIMR1)	<a href="#">on page 784</a>
0x002C	Channel Interrupt Mask Register (CIMR2)	<a href="#">on page 784</a>
0x0030	Watchdog Threshold Interrupt Status Register (WTISR)	<a href="#">on page 786</a>
0x0034	Watchdog Threshold Interrupt Mask Register (WTIMR)	<a href="#">on page 787</a>
0x0038...0x003F	Reserved	—
0x0040	DMA Enable Register (DMAE)	<a href="#">on page 788</a>
0x0044	DMA Channel Select Register 0 (DMAR0)	<a href="#">on page 789</a>
0x0048	DMA Channel Select Register 1 (DMAR1)	<a href="#">on page 789</a>
0x004C	DMA Channel Select Register 2 (DMAR2)	<a href="#">on page 789</a>
0x0050...0x005F	Reserved	—
0x0060	Threshold Register 0 (THRHLR0)	<a href="#">on page 791</a>
0x0064	Threshold Register 1 (THRHLR1)	<a href="#">on page 791</a>
0x0068	Threshold Register 2 (THRHLR2)	<a href="#">on page 791</a>
0x006C	Threshold Register 3 (THRHLR3)	<a href="#">on page 791</a>
0x0070...0x007F	Reserved	—
0x0080	Presampling Control Register (PSCR)	<a href="#">on page 793</a>
0x0084	Presampling Register 0 (PSR0)	<a href="#">on page 793</a>
0x0088	Presampling Register 1 (PSR1)	<a href="#">on page 793</a>
0x008C	Presampling Register 2 (PSR2)	<a href="#">on page 793</a>
0x0090...0x0093	Reserved	—
0x0094	Conversion Timing Register 0 (CTR0)	<a href="#">on page 795</a>
0x0098	Conversion Timing Register 1 (CTR1)	<a href="#">on page 795</a>
0x009C	Conversion Timing Register 2 (CTR2)	<a href="#">on page 795</a>
0x00A0...0x00A3	Reserved	—

**Table 381. 10-bit ADC\_0 digital registers (continued)**

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x00A4	Normal Conversion Mask Register 0 (NCMR0)	<a href="#">on page 796</a>
0x00A8	Normal Conversion Mask Register 1 (NCMR1)	<a href="#">on page 796</a>
0x00AC	Normal Conversion Mask Register 2 (NCMR2)	<a href="#">on page 796</a>
0x00B0...0x00B3	Reserved	—
0x00B4	Injected Conversion Mask Register 0 (JCMR0)	<a href="#">on page 798</a>
0x00B8	Injected Conversion Mask Register 1 (JCMR1)	<a href="#">on page 798</a>
0x00BC	Injected Conversion Mask Register 2 (JCMR2)	<a href="#">on page 798</a>
0x00C0...0x00C3	Reserved	—
0x00C4	Decode Signals Delay Register (DSDR)	<a href="#">on page 801</a>
0x00C8	Power-down Exit Delay Register (PDEDR)	<a href="#">on page 802</a>
0x00CC...0x00FF	Reserved	—
0x0100	Channel 0 Data Register (CDR0)	<a href="#">on page 802</a>
0x0104	Channel 1 Data Register (CDR1)	<a href="#">on page 802</a>
0x0108	Channel 2 Data Register (CDR2)	<a href="#">on page 802</a>
0x010C	Channel 3 Data Register (CDR3)	<a href="#">on page 802</a>
0x0110	Channel 4 Data Register (CDR4)	<a href="#">on page 802</a>
0x0114	Channel 5 Data Register (CDR5)	<a href="#">on page 802</a>
0x0118	Channel 6 Data Register (CDR6)	<a href="#">on page 802</a>
0x011C	Channel 7 Data Register (CDR7)	<a href="#">on page 802</a>
0x0120	Channel 8 Data Register (CDR8)	<a href="#">on page 802</a>
0x0124	Channel 9 Data Register (CDR9)	<a href="#">on page 802</a>
0x0128	Channel 10 Data Register (CDR10)	<a href="#">on page 802</a>
0x012C	Channel 11 Data Register (CDR11)	<a href="#">on page 802</a>
0x0130	Channel 12 Data Register (CDR12)	<a href="#">on page 802</a>
0x0134	Channel 13 Data Register (CDR13)	<a href="#">on page 802</a>
0x0138	Channel 14 Data Register (CDR14)	<a href="#">on page 802</a>
0x013C	Channel 15 Data Register (CDR15)	<a href="#">on page 802</a>
0x0140...0x017F	Reserved	—
0x0180	Channel 32 Data Register (CDR32)	<a href="#">on page 802</a>
0x0184	Channel 33 Data Register (CDR33)	<a href="#">on page 802</a>
0x0188	Channel 34 Data Register (CDR34)	<a href="#">on page 802</a>
0x018C	Channel 35 Data Register (CDR35)	<a href="#">on page 802</a>
0x0190	Channel 36 Data Register (CDR36)	<a href="#">on page 802</a>

Table 381. 10-bit ADC\_0 digital registers (continued)

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x0194	Channel 37 Data Register (CDR37)	<a href="#">on page 802</a>
0x0198	Channel 38 Data Register (CDR38)	<a href="#">on page 802</a>
0x019C	Channel 39 Data Register (CDR39)	<a href="#">on page 802</a>
0x01A0	Channel 40 Data Register (CDR40)	<a href="#">on page 802</a>
0x01A4	Channel 41 Data Register (CDR41)	<a href="#">on page 802</a>
0x01A8	Channel 42 Data Register (CDR42)	<a href="#">on page 802</a>
0x01AC	Channel 43 Data Register (CDR43)	<a href="#">on page 802</a>
0x01B0	Channel 44 Data Register (CDR44)	<a href="#">on page 802</a>
0x01B4	Channel 45 Data Register (CDR45)	<a href="#">on page 802</a>
0x01B8	Channel 46 Data Register (CDR46)	<a href="#">on page 802</a>
0x01BC	Channel 47 Data Register (CDR47)	<a href="#">on page 802</a>
0x01C0	Channel 48 Data Register (CDR48)	<a href="#">on page 802</a>
0x01C4	Channel 49 Data Register (CDR49)	<a href="#">on page 802</a>
0x01C8	Channel 50 Data Register (CDR50)	<a href="#">on page 802</a>
0x01CC	Channel 51 Data Register (CDR51)	<a href="#">on page 802</a>
0x01D0	Channel 52 Data Register (CDR52)	<a href="#">on page 802</a>
0x01D4	Channel 53 Data Register (CDR53)	<a href="#">on page 802</a>
0x01D8	Channel 54 Data Register (CDR54)	<a href="#">on page 802</a>
0x01DC	Channel 55 Data Register (CDR55)	<a href="#">on page 802</a>
0x01E0	Channel 56 Data Register (CDR56)	<a href="#">on page 802</a>
0x01E4	Channel 57 Data Register (CDR57)	<a href="#">on page 802</a>
0x01E8	Channel 58 Data Register (CDR58)	<a href="#">on page 802</a>
0x01EC	Channel 59 Data Register (CDR59)	<a href="#">on page 802</a>
0x01F0...0x01FF	Reserved	—
0x0200	Channel 64 Data Register (CDR64)	<a href="#">on page 802</a>
0x0204	Channel 65 Data Register (CDR65)	<a href="#">on page 802</a>
0x0208	Channel 66 Data Register (CDR66)	<a href="#">on page 802</a>
0x020C	Channel 67 Data Register (CDR67)	<a href="#">on page 802</a>
0x0210	Channel 68 Data Register (CDR68)	<a href="#">on page 802</a>
0x0214	Channel 69 Data Register (CDR69)	<a href="#">on page 802</a>
0x0218	Channel 70 Data Register (CDR70)	<a href="#">on page 802</a>
0x021C	Channel 71 Data Register (CDR71)	<a href="#">on page 802</a>
0x0220	Channel 72 Data Register (CDR72)	<a href="#">on page 802</a>

Table 381. 10-bit ADC\_0 digital registers (continued)

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x0224	Channel 73 Data Register (CDR73)	<a href="#">on page 802</a>
0x0228	Channel 74 Data Register (CDR74)	<a href="#">on page 802</a>
0x022C	Channel 75 Data Register (CDR75)	<a href="#">on page 802</a>
0x0230	Channel 76 Data Register (CDR76)	<a href="#">on page 802</a>
0x0234	Channel 77 Data Register (CDR77)	<a href="#">on page 802</a>
0x0238	Channel 78 Data Register (CDR78)	<a href="#">on page 802</a>
0x023C	Channel 79 Data Register (CDR79)	<a href="#">on page 802</a>
0x0240	Channel 80 Data Register (CDR80)	<a href="#">on page 802</a>
0x0244	Channel 81 Data Register (CDR81)	<a href="#">on page 802</a>
0x0248	Channel 82 Data Register (CDR82)	<a href="#">on page 802</a>
0x024C	Channel 83 Data Register (CDR83)	<a href="#">on page 802</a>
0x0250	Channel 84 Data Register (CDR84)	<a href="#">on page 802</a>
0x0254	Channel 85 Data Register (CDR85)	<a href="#">on page 802</a>
0x0258	Channel 86 Data Register (CDR86)	<a href="#">on page 802</a>
0x025C	Channel 87 Data Register (CDR87)	<a href="#">on page 802</a>
0x0260	Channel 88 Data Register (CDR88)	<a href="#">on page 802</a>
0x0264	Channel 89 Data Register (CDR89)	<a href="#">on page 802</a>
0x0268	Channel 90 Data Register (CDR90)	<a href="#">on page 802</a>
0x026C	Channel 91 Data Register (CDR91)	<a href="#">on page 802</a>
0x0270	Channel 92 Data Register (CDR92)	<a href="#">on page 802</a>
0x0274	Channel 93 Data Register (CDR93)	<a href="#">on page 802</a>
0x0278	Channel 94 Data Register (CDR94)	<a href="#">on page 802</a>
0x027C	Channel 95 Data Register (CDR95)	<a href="#">on page 802</a>
0x0280	Threshold Register 4 (THRHLR4)	<a href="#">on page 802</a>
0x0284	Threshold Register 5 (THRHLR5)	<a href="#">on page 802</a>
0x0288...0x02AF	Reserved	—
0x02B0	Channel Watchdog Selection Register 0 (CWSEL0)	<a href="#">on page 804</a>
0x02B4	Channel Watchdog Selection Register 1 (CWSEL1)	<a href="#">on page 804</a>
0x02B8...0x02BF	Reserved	—
0x02C0	Channel Watchdog Selection Register 4 (CWSEL4)	<a href="#">on page 804</a>
0x02C4	Channel Watchdog Selection Register 5 (CWSEL5)	<a href="#">on page 804</a>
0x02C8	Channel Watchdog Selection Register 6 (CWSEL6)	<a href="#">on page 804</a>
0x02CC	Channel Watchdog Selection Register 7 (CWSEL7)	<a href="#">on page 804</a>

**Table 381. 10-bit ADC\_0 digital registers (continued)**

Base address: 0xFFE0_0000		Location
Address offset	Register name	
0x02D0	Channel Watchdog Selection Register 8 (CWSEL8)	<a href="#">on page 804</a>
0x02D4	Channel Watchdog Selection Register 9 (CWSEL9)	<a href="#">on page 804</a>
0x02D8	Channel Watchdog Selection Register 10 (CWSEL10)	<a href="#">on page 804</a>
0x02DC	Channel Watchdog Selection Register 11 (CWSEL11)	<a href="#">on page 804</a>
0x02E0	Channel Watchdog Enable Register 0 (CWENR0)	<a href="#">on page 811</a>
0x02E4	Channel Watchdog Enable Register 1 (CWENR1)	<a href="#">on page 811</a>
0x02E8	Channel Watchdog Enable Register 2 (CWENR2)	<a href="#">on page 811</a>
0x02EC...0x02EF	Reserved	—
0x02F0	Analog Watchdog Out of Range register 0 (AWORR0)	<a href="#">on page 814</a>
0x02F4	Analog Watchdog Out of Range register 1 (AWORR1)	<a href="#">on page 814</a>
0x02F8	Analog Watchdog Out of Range register 2 (AWORR2)	<a href="#">on page 814</a>
0x2FC...0x02FF	Reserved	—

[Table 382](#) lists the ADC\_1 registers with their address offsets and reset values.

**Table 382. 12-bit ADC\_1 digital registers**

Base address: 0xFFE0_4000		Location
Address offset	Register name	
0x0000	Main Configuration Register (MCR)	<a href="#">on page 777</a>
0x0004	Main Status Register (MSR))	<a href="#">on page 779</a>
0x0008...0x000F	Reserved	—
0x0010	Interrupt Status Register (ISR)	<a href="#">on page 781</a>
0x0014	Channel Pending Register (CEOCFR0)	<a href="#">on page 781</a>
0x001C	Channel Pending Register (CEOCFR1)	<a href="#">on page 781</a>
0x0020	Interrupt Mask Register (IMR)	<a href="#">on page 783</a>
0x0024	Channel Interrupt Mask Register (CIMR0)	<a href="#">on page 784</a>
0x0028	Channel Interrupt Mask Register (CIMR1)	<a href="#">on page 784</a>
0x0030	Watchdog Threshold Interrupt Status Register (WTISR)	<a href="#">on page 786</a>
0x0034	Watchdog Threshold Interrupt Mask Register (WTIMR)	<a href="#">on page 787</a>
0x0038...0x003F	Reserved	—
0x0040	DMA Enable Register (DMAE)	<a href="#">on page 788</a>
0x0044	DMA Channel Select Register 0 (DMAR0)	<a href="#">on page 788</a>
0x0048	DMA Channel Select Register 1 (DMAR1)	<a href="#">on page 788</a>
0x0060	Threshold Register 0 (THRHLR0)	<a href="#">on page 791</a>

**Table 382. 12-bit ADC\_1 digital registers (continued)**

Base address: 0xFFE0_4000		Location
Address offset	Register name	
0x0064	Threshold Register 1 (THRHLR1)	<a href="#">on page 791</a>
0x0068	Threshold Register 2 (THRHLR2)	<a href="#">on page 791</a>
0x006C...0x007F	Reserved	—
0x0080	Presampling Control Register (PSCR)	<a href="#">on page 793</a>
0x0084	Presampling Register 0 (PSR0)	<a href="#">on page 793</a>
0x0088	Presampling Register 1(PSR1)	<a href="#">on page 793</a>
0x008C...0x0093	Reserved	—
0x0094	Conversion Timing Register 0 (CTR0)	<a href="#">on page 795</a>
0x0098	Conversion Timing Register 1(CTR1)	<a href="#">on page 795</a>
0x00A4	Normal Conversion Mask Register 0 (NCMR0)	<a href="#">on page 796</a>
0x00A8	Normal Conversion Mask Register 1(NCMR1)	<a href="#">on page 796</a>
0x00B4	Injected Conversion Mask Register 0 (JCMR0)	<a href="#">on page 798</a>
0x00B8	Injected Conversion Mask Register 1(JCMR1)	<a href="#">on page 798</a>
0x00BC...0x00C7	Reserved	—
0x00C8	Power-down Exit Delay Register (PDED R)	<a href="#">on page 802</a>
0x00CC...0x00FF	Reserved	—
0x0100	Channel 0 Data Register (CDR0)	<a href="#">on page 802</a>
0x0104	Channel 1 Data Register (CDR1)	<a href="#">on page 802</a>
0x0108	Channel 2 Data Register (CDR2)	<a href="#">on page 802</a>
0x010C	Channel 3 Data Register (CDR3)	<a href="#">on page 802</a>
0x0110	Channel 4 Data Register (CDR4)	<a href="#">on page 802</a>
0x0114	Channel 5 Data Register (CDR5)	<a href="#">on page 802</a>
0x0118	Channel 6 Data Register (CDR6)	<a href="#">on page 802</a>
0x011C	Channel 7 Data Register (CDR7)	<a href="#">on page 802</a>
0x0120	Channel 8 Data Register (CDR8)	<a href="#">on page 802</a>
0x0124	Channel 9 Data Register (CDR9)	<a href="#">on page 802</a>
0x0128	Channel 10 Data Register (CDR10)	<a href="#">on page 802</a>
0x012C	Channel 11 Data Register (CDR11)	<a href="#">on page 802</a>
0x0130	Channel 12 Data Register (CDR12)	<a href="#">on page 802</a>
0x0134	Channel 13 Data Register (CDR13)	<a href="#">on page 802</a>
0x0138	Channel 14 Data Register (CDR14)	<a href="#">on page 802</a>
0x013C	Channel 15 Data Register (CDR15)	<a href="#">on page 802</a>
0x0140...0x017F	Reserved	—



Table 382. 12-bit ADC\_1 digital registers (continued)

Base address: 0xFFE0_4000		Location
Address offset	Register name	
0x0180	Channel 32 Data Register (CDR32)	<a href="#">on page 802</a>
0x0184	Channel 33 Data Register (CDR33)	<a href="#">on page 802</a>
0x0188	Channel 34 Data Register (CDR34)	<a href="#">on page 802</a>
0x018C	Channel 35 Data Register (CDR35)	<a href="#">on page 802</a>
0x0190	Channel 36 Data Register (CDR36)	<a href="#">on page 802</a>
0x0194	Channel 37 Data Register (CDR37)	<a href="#">on page 802</a>
0x0198	Channel 38 Data Register (CDR38)	<a href="#">on page 802</a>
0x019C	Channel 39 Data Register (CDR39)	<a href="#">on page 802</a>
0x01A0...0x02AF	Reserved	—
0x02B0	Channel Watchdog Selection Register 0 (CWSEL0)	<a href="#">on page 804</a>
0x02B4	Channel Watchdog Selection Register 1 (CWSEL1)	<a href="#">on page 804</a>
0x02B8...0x02BF	Reserved	—
0x02C0	Channel Watchdog Selection Register 4 (CWSEL4)	<a href="#">on page 804</a>
0x02C4...0x02DF	Reserved	—
0x02E0	Channel Watchdog Enable Register 0 (CWENR0)	<a href="#">on page 811</a>
0x02E4	Channel Watchdog Enable Register 1 (CWENR1)	<a href="#">on page 811</a>
0x02F0	Analog Watchdog Out of Range register 0 (AWORR0)	<a href="#">on page 814</a>
0x02F4	Analog Watchdog Out of Range register 1 (AWORR1)	<a href="#">on page 814</a>
0x02F8...0x02FF	Reserved	—

## 28.4.2 Control logic registers

### 28.4.2.1 Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Address: Base + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OWREN	WLSIDE	MODE	0	0	0	0	NSTART	0	JTRGEN	JEDGE	JSTART	0	0	CTUEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	ADCLK SEL	ABORT CHAIN	ABORT	ACKO	0	0	0	0	PWDN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 423. Main Configuration Register (MCR)

Table 383. MCR field descriptions

Field	Description
OWREN	Overwrite enable This bit enables or disables the functionality to overwrite unread converted data. 0 Prevents overwrite of unread converted data; new result is discarded 1 Enables converted data to be overwritten by a new conversion
WLSIDE	Write left/right-aligned 0 The conversion data is written right-aligned. 1 Data is left-aligned (from 15 to (15 – resolution + 1)). The WLSIDE bit affects all the CDR registers simultaneously. See <a href="#">Figure 462</a> and <a href="#">Figure 463</a> .
MODE	One Shot/Scan 0 One Shot Mode—Configures the normal conversion of one chain. 1 Scan Mode—Configures continuous chain conversion mode; when the programmed chain conversion is finished it restarts immediately.
NSTART	Normal Start conversion Setting this bit starts the chain or scan conversion. Resetting this bit during scan mode causes the current chain conversion to finish, then stops the operation. This bit stays high while the conversion is ongoing (or pending during injection mode). 0 Causes the current chain conversion to finish and stops the operation 1 Starts the chain or scan conversion
JTRGEN	Injection external trigger enable 0 External trigger disabled for channel injection 1 External trigger enabled for channel injection
JEDGE	Injection trigger edge selection Edge selection for external trigger, if JTRGEN = 1. 0 Selects falling edge for the external trigger 1 Selects rising edge for the external trigger
JSTART	Injection start Setting this bit will start the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.

**Table 383. MCR field descriptions (continued)**

Field	Description
CTUEN	Cross trigger unit conversion enable 0 CTU triggered conversion disabled 1 CTU triggered conversion enabled
ADCLKSEL	Analog clock select This bit can only be written when ADC in Power-Down mode 0 ADC clock frequency is half Peripheral Set Clock frequency 1 ADC clock frequency is equal to Peripheral Set Clock frequency
ABORTCHAIN	Abort Chain When this bit is set, the ongoing Chain Conversion is aborted. This bit is reset by hardware as soon as a new conversion is requested. 0 Conversion is not affected 1 Aborts the ongoing chain conversion
ABORT	Abort Conversion When this bit is set, the ongoing conversion is aborted and a new conversion is invoked. This bit is reset by hardware as soon as a new conversion is invoked. If it is set during a scan chain, only the ongoing conversion is aborted and the next conversion is performed as planned. 0 Conversion is not affected 1 Aborts the ongoing conversion
ACKO	Auto-clock-off enable If set, this bit enables the Auto clock off feature. 0 Auto clock off disabled 1 Auto clock off enabled
PWDN	Power-down enable When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, resetting this bit starts ADC transition to IDLE mode. 0 ADC is in normal mode 1 ADC has been requested to power down

### 28.4.2.2 Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Address: Base + 0x0004

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	NSTART	JABORT	0	0	JSTART	0	0	0	CTUSTART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CHADDR								0	0	0	ACK0	0	0	ADCSTATUS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 424. Main Status Register (MSR)

Table 384. MSR field descriptions

Field	Description
NSTART	This status bit is used to signal that a Normal conversion is ongoing.
JABORT	This status bit is used to signal that an Injected conversion has been aborted. This bit is reset when a new injected conversion starts.
JSTART	This status bit is used to signal that an Injected conversion is ongoing.
CTUSTART	This status bit is used to signal that a CTU conversion is ongoing.
CHADDR	Current conversion channel address This status field indicates current conversion channel address.
ACK0	Auto-clock-off enable This status bit is used to signal if the Auto-clock-off feature is on.
ADCSTATUS	The value of this parameter depends on ADC status: 000 IDLE — The ADC is powered up but idle. 001 Power-down — The ADC is powered down. 010 Wait state — The ADC is waiting for an external multiplexer. This occurs only when the DSDR register is non-zero. 011 Reserved 100 Sample — The ADC is sampling the analog signal. 101 Reserved 110 Conversion — The ADC is converting the sampled signal. 111 Reserved

Note: *MSR[JSTART] is automatically set when the injected conversion starts. At the same time MCR[JSTART] is reset, allowing the software to program a new start of conversion.*  
*The JCMR registers do not change their values.*

### 28.4.3 Interrupt registers

#### 28.4.3.1 Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

Address: Base + 0x0010

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	EO CTU	JEOC	JECH	EOC	ECH
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 425. Interrupt Status Register (ISR)

Table 385. ISR field descriptions

Field	Description
EOCTU	End of CTU Conversion interrupt flag When this bit is set, an EOCTU interrupt has occurred.
JEOC	End of Injected Channel Conversion interrupt flag When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt flag When this bit is set, a JECH interrupt has occurred.
EOC	End of Channel Conversion interrupt flag When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt flag When this bit is set, an ECH interrupt has occurred.

#### 28.4.3.2 Channel Pending Registers (CEOCFR[0...2])

Table 386 shows the exact number of available channels.

Table 386. CEOCFR[0...2] register description

Register	Description	ADC
CEOCFR0	End of conversion pending interrupt for channel 0 to 15 (precision channels)	ADC_0
		ADC_1

Table 386. CEOCFR[0...2] register description (continued)

Register	Description	ADC
CEOCFR1	End of conversion pending interrupt for channel 32 to 59 (standard channels)	ADC_0
	End of conversion pending interrupt for channel 32 to 39 (standard channels)	ADC_1
CEOCFR2	End of conversion pending interrupt for channel 64 to 95 (external multiplexed channels)	ADC_0

Address: Base + 0x0014

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC_CH15	EOC_CH14	EOC_CH13	EOC_CH12	EOC_CH11	EOC_CH10	EOC_CH9	EOC_CH8	EOC_CH7	EOC_CH6	EOC_CH5	EOC_CH4	EOC_CH3	EOC_CH2	EOC_CH1	EOC_CH0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 426. Channel Pending Register 0 (CEOCFR0)

Address: Base + 0x0018

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_CH59	EOC_CH58	EOC_CH57	EOC_CH56	EOC_CH55	EOC_CH54	EOC_CH53	EOC_CH52	EOC_CH51	EOC_CH50	EOC_CH49	EOC_CH48
W					w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC_CH47	EOC_CH46	EOC_CH43	EOC_CH44	EOC_CH43	EOC_CH42	EOC_CH41	EOC_CH40	EOC_CH39	EOC_CH38	EOC_CH37	EOC_CH36	EOC_CH35	EOC_CH34	EOC_CH33	EOC_CH32
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 427. Channel Pending Register 1 (CEOCFR1) - ADC\_0

Address: Base + 0x0018

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	EOC_CH39	EOC_CH38	EOC_CH37	EOC_CH36	EOC_CH35	EOC_CH34	EOC_CH33	EOC_CH32
W									w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 428. Channel Pending Register 1 (CEOCFR1) - ADC\_1

Address: Base + 0x001C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EOC_CH95	EOC_CH94	EOC_CH93	EOC_CH92	EOC_CH91	EOC_CH90	EOC_CH89	EOC_CH88	EOC_CH87	EOC_CH86	EOC_CH85	EOC_CH84	EOC_CH83	EOC_CH82	EOC_CH81	EOC_CH80
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC_CH79	EOC_CH78	EOC_CH77	EOC_CH76	EOC_CH75	EOC_CH74	EOC_CH73	EOC_CH72	EOC_CH71	EOC_CH70	EOC_CH69	EOC_CH68	EOC_CH67	EOC_CH66	EOC_CH65	EOC_CH64
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 429. Channel Pending Register 2 (CEOCFR2)

Table 387. CEOCFR field descriptions

Field	Description
EOC_CHn	When set, the measure of channel n is completed.

### 28.4.3.3 Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.



Address: Base + 0x0020

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	MSKE OCTU	MSK JEOC	MSK JECH	MSK EOC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 430. Interrupt Mask Register (IMR)

Table 388. Interrupt Mask Register (IMR) field descriptions

Field	Description
MSKEOCTU	Mask for end of CTU conversion (EOCTU) interrupt When set, the EOCTU interrupt is enabled.
MSKJEOC	Mask for end of injected channel conversion (JEOC) interrupt When set, the JEOC interrupt is enabled.
MSKJECH	Mask for end of injected chain conversion (JECH) interrupt When set, the JECH interrupt is enabled.
MSKEOC	Mask for end of channel conversion (EOC) interrupt When set, the EOC interrupt is enabled.
MSKECH	Mask for end of chain conversion (ECH) interrupt When set, the ECH interrupt is enabled.

28.4.3.4 Channel Interrupt Mask Register (CIMR[0...2])

Table 389 shows the exact number of available channels.

Table 389. CIMR[0...2] register description

Register	Description	ADC
CIMR0	Enable bit for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
CIMR1	Enable bit for channel 32 to 59 (standard channels)	ADC_0
	Enable bit for channel 32 to 39 (standard channels)	ADC_1
CIMR2	Enable bit for channel 64 to 95 (external multiplexed channels)	ADC_0



Address: Base + 0x0024

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 431. Channel Interrupt Mask Register 0 (CIMR0)

Address: Base + 0x0028

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM
W					59	58	57	56	55	54	53	52	51	50	49	48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM
W	47	46	43	44	43	42	41	40	39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 432. Channel Interrupt Mask Register 1 (CIMR1) - ADC\_0

Address: Base + 0x0028

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM
W									39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 433. Channel Interrupt Mask Register 1 (CIMR1) - ADC\_1

Address: Base + 0x002C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM
W	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM	CIM
W	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 434. Channel Interrupt Mask Register 2 (CIMR2)

Table 390. CIMR field descriptions

Field	Description
CIMn	Interrupt enable When set (CIMn = 1), interrupt for channel n is enabled.

28.4.3.5 Watchdog Threshold Interrupt Status Register (WTISR)

For ADC\_0 (10-bit)

Address: Base + 0x0030

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	WDG	WDG	WDG	WDG	WDG	WDG	WDG	WDG	WDG	WDG	WDG	WDG
W					w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 435. ADC\_0 Watchdog Threshold Interrupt Status Register (WTISR)

Table 391. ADC\_0 WTISR field descriptions

Field	Description
WDGxH	This corresponds to the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0...5]).
WDGxL	This corresponds to the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0...5]).



For ADC\_1 (12-bit)

Address: Base + 0x0030

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	WDG 2H	WDG 2L	WDG 1H	WDG 1L	WDG 0H	WDG 0L
W											w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 436. ADC\_1 Watchdog Threshold Interrupt Status Register (WTISR)

Table 392. ADC\_1 WTISR field descriptions

Field	Description
WDGxH	This corresponds to the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0...2]).
WDGxL	This corresponds to the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0...2]).

28.4.3.6 Watchdog Threshold Interrupt Mask Register (WTIMR)

For ADC\_0 (10-bit)

Address: Base + 0x0034

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	MSK WDG 5H	MSK WDG 5L	MSK WDG 4H	MSK WDG 4L	MSK WDG 3H	MSK WDG 3L	MSK WDG 2H	MSK WDG 2L	MSK WDG 1H	MSK WDG 1L	MSK WDG 0H	MSK WDG 0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 437. ADC\_0 Watchdog Threshold Interrupt Mask Register (WTIMR)

**Table 393. ADC\_0 WTIMR field descriptions**

Field	Description
2x + 1	MSKWDGxH [x = 0...5] This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold. When set the interrupt is enabled.
2x	MSKWDGxL [x = 0...5]his corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold. When set the interrupt is enabled.

For ADC\_1 (12-bit)

Address: Base + 0x0034

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	MSK WDG 2H	MSK WDG 2L	MSK WDG 1H	MSK WDG 1L	MSK WDG 0H	MSK WDG 0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 438. ADC\_1 Watchdog Threshold Interrupt Mask Register (WTIMR)**

**Table 394. ADC\_1 WTIMR field descriptions**

Field	Description
MSKWDGxH	This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0...2]). When set the interrupt is enabled.
MSKWDGxL	This corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0...2]). When set the interrupt is enabled.

**28.4.4 DMA registers**

**28.4.4.1 DMA Enable Register (DMAE)**

The DMA Enable (DMAE) register sets up the DMA for use with the ADC.

Address: Base + 0x0040

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DCLR	DMAEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 439. DMA Enable Register (DMAE)

Table 395. DMAE field descriptions

Field	Description
DCLR	DMA clear sequence enable 0 DMA request cleared by Acknowledge from DMA controller 1 DMA request cleared on read of data registers
DMAEN	DMA global enable 0 DMA feature disabled 1 DMA feature enabled

28.4.4.2 DMA Channel Select Register (DMAR[0...2])

Table 396 shows the exact number of available channels.

Table 396. DMAR[0...2] register description

Register	Description	ADC
DMAR0	Enable bits for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
DMAR1	Enable bits for channel 32 to 59 (standard channels)	ADC_0
	Enable bits for channel 32 to 39 (standard channels)	ADC_1
DMAR2	Enable bits for channel 64 to 95 (external multiplexed channels)	ADC_0

Address: Base + 0x0044

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 440. DMA Channel Select Register 0 (DMAR0)

Address: Base + 0x0048

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA
W					59	58	57	56	55	54	53	52	51	50	49	48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA
W	47	46	43	44	43	42	41	40	39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 441. DMA Channel Select Register 1 (DMAR1) - ADC\_0

Address: Base + 0x0048

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA
W									39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 442. DMA Channel Select Register 1 (DMAR1) - ADC\_1

Address: Base + 0x004C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA
W	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA	DMA
W	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 443. DMA Channel Select Register 2 (DMAR2)

Table 397. DMARx field descriptions

Field	Description
DMA <sub>n</sub>	DMA enable When set (DMA <sub>n</sub> = 1), channel n is enabled to transfer data in DMA mode.

### 28.4.5 Threshold registers

#### 28.4.5.1 Threshold Register (THRHLR)

For ADC\_0 (10-bit): THRHLR[0...5]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	THRH									
W																
Reset	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

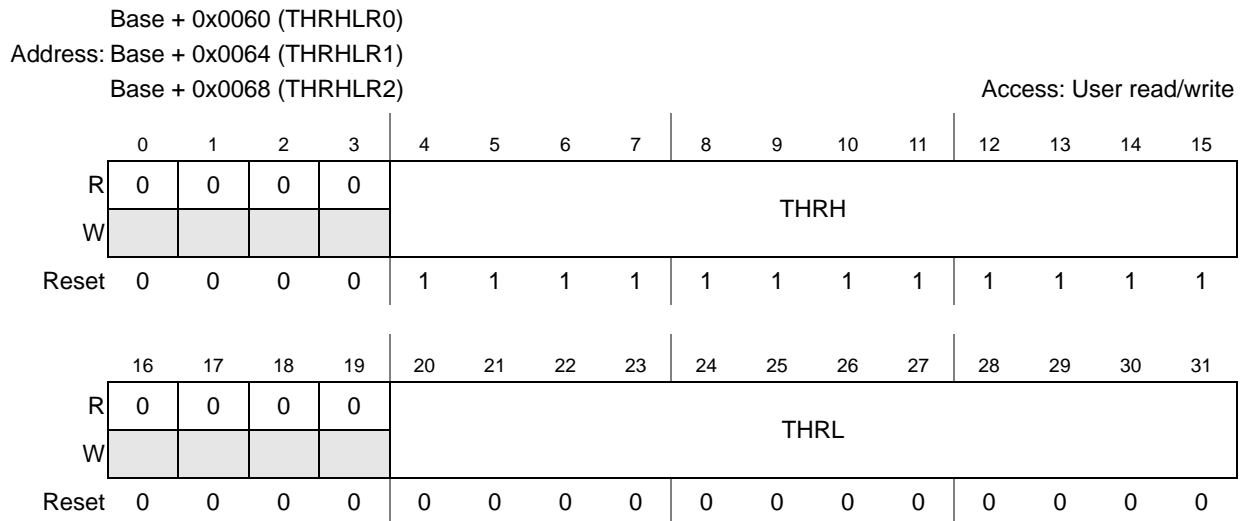
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	THRL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 444. ADC\_0 Threshold Register THRHLR[0...5]

**Table 398. ADC\_0 THRHLR field descriptions**

Field	Description
THRH	High threshold value for channel <i>n</i> .
THRL	Low threshold value for channel <i>n</i> .

For ADC\_1 (12-bit): THRHLR[0...2]



**Figure 445. ADC\_1 Threshold Register THRHLR[0...2]**

**Table 399. ADC\_1 THRHLR field descriptions**

Field	Description
THRH	High threshold value for channel <i>n</i> .
THRL	Low threshold value for channel <i>n</i> .



## 28.4.6 Presampling registers

### 28.4.6.1 Presampling Control Register (PSCR)

Address: Base + 0x0080

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
R	0	0	0	0	0	0	0	0	0	PREVAL2			PREVAL1			PREVAL0		PRE CON V
W																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Figure 446. Presampling Control Register (PSCR)

Table 400. PSCR field descriptions

Field	Description
PREVAL2	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (external multiplexed channels). See <a href="#">Table 379</a> .
PREVAL1	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (standard channels). See <a href="#">Table 379</a> .
PREVAL0	Internal voltage selection for presampling Selects analog input voltage for presampling from the available two internal voltages (precision channels). See <a href="#">Table 379</a> .
PRECONV	Convert presampled value If bit PRECONV is set, presampling is followed by the conversion. Sampling will be bypassed and conversion of presampled data will be done.

### 28.4.6.2 Presampling Register (PSR[0...2])

[Table 401](#) shows the exact number of available channels.

Table 401. PSR[0...2] register description

Register	Description	ADC
PSR0	Enable bits of presampling for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
PSR1	Enable bits of presampling for channel 32 to 59 (standard channels)	ADC_0
	Enable bits of presampling for channel 32 to 39 (standard channels)	ADC_1
PSR2	Enable bits of presampling for channel 64 to 95 (external multiplexed channels)	ADC_0

Address: Base + 0x0084

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 447. Presampling Register 0 (PSR0)

Address: Base + 0x0088

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W					59	58	57	56	55	54	53	52	51	50	49	48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W	47	46	43	44	43	42	41	40	39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 448. Presampling Register 1 (PSR1) - ADC\_0

Address: Base + 0x0088

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W									39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 449. Presampling Register 1 (PSR1) - ADC\_1

Address: Base + 0x008C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES	PRES
W	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 450. Presampling Register 2 (PSR2)

Table 402. PSR field descriptions

Field	Description
PRESn	Presampling enable When set (PRESn = 1), presampling is enabled for channel n.

### 28.4.7 Conversion timing registers CTR[0...2]

Table 403 shows the exact number of available channels.

Table 403. CTR[0...2] register description

Register	Description	ADC
CTR0	Associated to internal precision channels (from 0 to 15)	ADC_0
		ADC_1
CTR1	Associated to internal standard channels (from 32 to 59)	ADC_0
	Associated to internal standard channels (from 32 to 39)	ADC_1
CTR2	Associated to external multiplexed channels (from 64 to 95)	ADC_0

Base + 0x0094 (CTR0)  
 Address: Base + 0x0098 (CTR1) Access: User read/write  
 Base + 0x009C (CTR2)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPLATCH	0	OFFSHIFT (1)		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1

Figure 451. Conversion timing registers CTR[0...2]

1. Available only on CTR0

Table 404. CTR field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration
OFFSHIFT	Configuration for offset shift characteristic 00 No shift (that is the transition between codes 000h and 001h) is reached when the $A_{VIN}$ (analog input voltage) is equal to 1 LSB. 01 Transition between code 000h and 001h is reached when the $A_{VIN}$ is equal to 1/2 LSB 10 Transition between code 00h and 001h is reached when the $A_{VIN}$ is equal to 0 11 Not used <b>Note:</b> Available only on CTR0
INPCMP	Configuration bits for comparison phase duration
INPSAMP	Configuration bits for sampling phase duration

### 28.4.8 Mask registers

#### 28.4.8.1 Introduction

These registers are used to program which of the 96 input channels must be converted during Normal and Injected conversion.

#### 28.4.8.2 Normal Conversion Mask Registers (NCMR[0...2])

Table 405 shows the exact number of available channels.

Table 405. NCMR[0..2] register description

Register	Description	ADC
NCMR0	Enable bits of normal sampling for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
NCMR1	Enable bits of normal sampling for channel 32 to 59 (standard channels)	ADC_0
	Enable bits of normal sampling for channel 32 to 39 (standard channels)	ADC_1
NCMR2	Enable bits of normal sampling for channel 64 to 95 (external multiplexed channels)	ADC_0

Address: Base + 0x00A4

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 452. Normal Conversion Mask Register 0 (NCMR0)

Address: Base + 0x00A8

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	CH59	CH58	CH57	CH56	CH55	CH54	CH53	CH52	CH51	CH50	CH49	CH48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 453. Normal Conversion Mask Register 1 (NCMR1) - ADC\_0

Address: Base + 0x00A8

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 454. Normal Conversion Mask Register 1 (NCMR1) - ADC\_1

Address: Base + 0x00AC

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 455. Normal Conversion Mask Register 2 (NCMR2)

Table 406. NCMR field descriptions

Field	Description
CHn	Sampling enable When set Sampling is enabled for channel n.

Note: The implicit channel conversion priority in the case in which all channels are selected is the following: ADCn\_P[0:x], ADCn\_S[0:y], ADCn\_X[0:z].

The channels always start with 0, the lowest index.

### 28.4.8.3 Injected Conversion Mask Registers (JCMR[0...2])

Table 407 shows the exact number of available channels.

Table 407. JCMR[0...2] register description

Register	Description	ADC
JCMR0	Enable bits of injected sampling for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
JCMR1	Enable bits of injected sampling for channel 32 to 59 (standard channels)	ADC_0
	Enable bits of injected sampling for channel 32 to 39 (standard channels)	ADC_1
JCMR2	Enable bits of injected sampling for channel 64 to 95 (external multiplexed channels)	ADC_0

Address: Base + 0x00B4

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 456. Injected Conversion Mask Register 0 (JCMR0)

Address: Base + 0x00B8

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	CH59	CH58	CH57	CH56	CH55	CH54	CH53	CH52	CH51	CH50	CH49	CH48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH47	CH46	CH45	CH44	CH43	CH42	CH41	CH40	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 457. Injected Conversion Mask Register 1 (JCMR1) - ADC\_0

Address: Base + 0x00B8

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CH39	CH38	CH37	CH36	CH35	CH34	CH33	CH32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 458. Injected Conversion Mask Register 1 (JCMR1) - ADC\_1

Address: Base + 0x00BC

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CH95	CH94	CH93	CH92	CH91	CH90	CH89	CH88	CH87	CH86	CH85	CH84	CH83	CH82	CH81	CH80
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CH79	CH78	CH77	CH76	CH75	CH74	CH73	CH72	CH71	CH70	CH69	CH68	CH67	CH66	CH65	CH64
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 459. Injected Conversion Mask Register 2 (JCMR2)

Table 408. JCMR field descriptions

Field	Description
CHn	Sampling enable When set, sampling is enabled for channel n.



### 28.4.9 Delay registers

#### 28.4.9.1 Decode Signals Delay Register (DSDR)<sup>(y)</sup>

Address: Base + 0x00C4

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	DSD											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 460. Decode Signals Delay Register (DSDR)

Table 409. DSDR field descriptions

Field	Description
DSD	<p>Delay between the external decode signals and the start of the sampling phase It is used to take into account the settling time of the external multiplexer. The decode signal delay is calculated as: <math>DSD \times 1/\text{frequency of ADC clock}</math>.</p> <p><b>Note:</b> when <math>\text{ADC clock} = \text{Peripheral Clock}/2</math> the DSD has to be incremented by 2 to see an additional ADC clock cycle delay on the decode signal. For example: DSD = 0; 0 ADC clock cycle delay DSD = 2; 1 ADC clock cycle delay DSD = 4; 2 ADC clock cycles delay</p>

y. Implemented only on ADC\_0

**28.4.9.2 Power-down Exit Delay Register (PDEDR)**

Address: Base + 0x00C8

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PDED							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 461. Power-down Exit Delay Register (PDEDR)**

**Table 410. PDEDR field descriptions**

Field	Description
PDED	Delay between the power-down bit reset and the start of conversion. The delay is to allow time for the ADC power supply to settle before commencing conversions. The power down delay is calculated as: PDED x 1/frequency of ADC clock.

**28.4.10 Data registers**

**28.4.10.1 Introduction**

ADC conversion results are stored in data registers. There is one register per channel.

**28.4.10.2 Channel Data Register (CDR[0...95])**

*Table 411* shows the exact number of available channels.

**Table 411. CDR[0...95] register description**

Registers	Description	ADC
CDR0[0...15]	Enable bits of injected sampling for channel 0 to 15 (precision channels)	ADC_0
CDR1[0...15]		ADC_1
CDR0[32...59]	Enable bits of injected sampling for channel 32 to 59 (standard channels)	ADC_0
CDR1[32...59]	Enable bits of injected sampling for channel 32 to 39 (standard channels)	ADC_1
CDR0[64...95]	Enable bits of injected sampling for channel 64 to 95 (external multiplexed channels)	ADC_0

Each data register also gives information regarding the corresponding result as described below.

For ADC\_0 10-bit:

Address: See [Table 381](#)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CDATA[0:9] (MCR[WLSIDE] = 0)									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	CDATA[0:9] (MCR[WLSIDE] = 1)											0	0	0	0	0	0
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 462. Channel Data Register (CDR[0...95])

For ADC\_1 12-bit:

Address: See [Table 382](#)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT [0:1]	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	CDATA[0:11] (MCR[WLSIDE] = 0)											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDATA[0:11] (MCR[WLSIDE] = 1)											0	0	0	0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 463. Channel Data Register (CDR[0...95])

**Table 412. CDR field descriptions**

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	<p>Overwrite data</p> <p>This bit signals that the previous converted data has been overwritten by a new conversion. This functionality depends on the value of MCR[OWREN]:</p> <ul style="list-style-type: none"> <li>– When OWREN = 0, then OVERW is frozen to 0 and CDATA field is protected against being overwritten until being read.</li> <li>– When OWREN = 1, then OVERW flags the CDATA field overwrite status.</li> </ul> <p>0 Converted data has not been overwritten                      1 Previous converted data has been overwritten before having been read</p>
RESULT	<p>This bit reflects the mode of conversion for the corresponding channel.</p> <p>00 Data is a result of Normal conversion mode                      01 Data is a result of Injected conversion mode                      10 Data is a result of CTU conversion mode                      11 Reserved</p>
CDATA	Channel 0-95 converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this bitfield can be changed as shown in <a href="#">Figure 462</a> and <a href="#">Figure 463</a> .

**28.4.11 Watchdog register**

**28.4.11.1 Channel Watchdog Select Register (CWSELR[0...11])**

Register WSEL\_CHn[3:0] = Selects the threshold register which provides the values to be used for upper and lower bounds for channel n.

[Table 413](#) shows the exact number of available channels.

**Table 413. CWSELR[0...11] register description**

Register	Description	ADC
CWSELR[0...1]	Channel watchdog select register for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
CWSELR[4...7]	Channel watchdog select register for channel 32 to 59 (standard channels)	ADC_0
CWSELR[4]	Channel watchdog select register for channel 32 to 39 (standard channels)	ADC_1
CWSELR[8...11]	Channel watchdog select register for channel 64 to 95 (external multiplexed channels)	ADC_0

Offset: 0x02B0

Access: User read/write

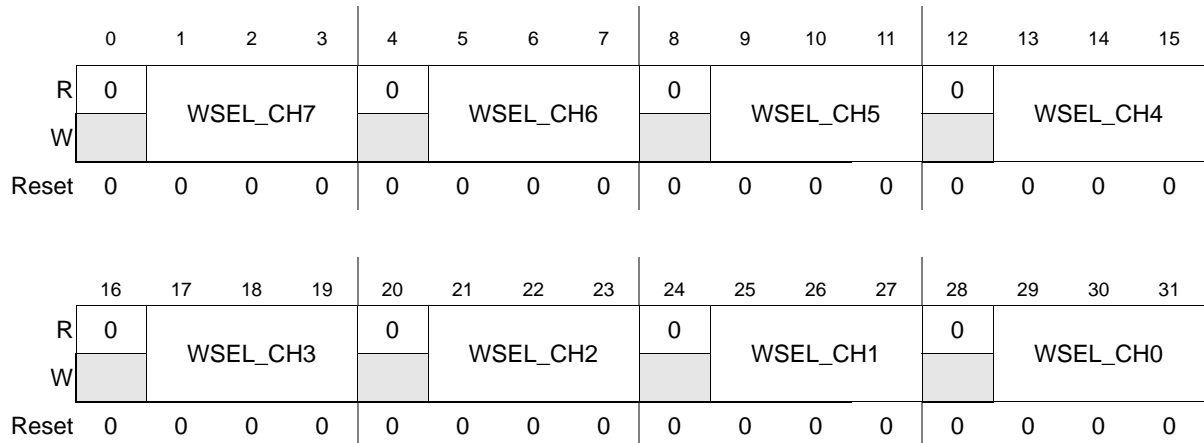


Figure 464. Channel Watchdog Select Register 0 (CWSELR0) – ADC\_0

Table 414. CWSELR field descriptions – ADC\_0

Field	Description
WSEL_CHn	Channel Watchdog select for channel n 000: THRHLR0 register is selected 001: THRHLR1 register is selected 010: THRHLR2 register is selected 011: THRHLR3 register is selected 100: THRHLR4 register is selected 101: THRHLR5 register is selected 110: RESERVED 111: RESERVED

Offset: 0x02B4

Access: User read/write

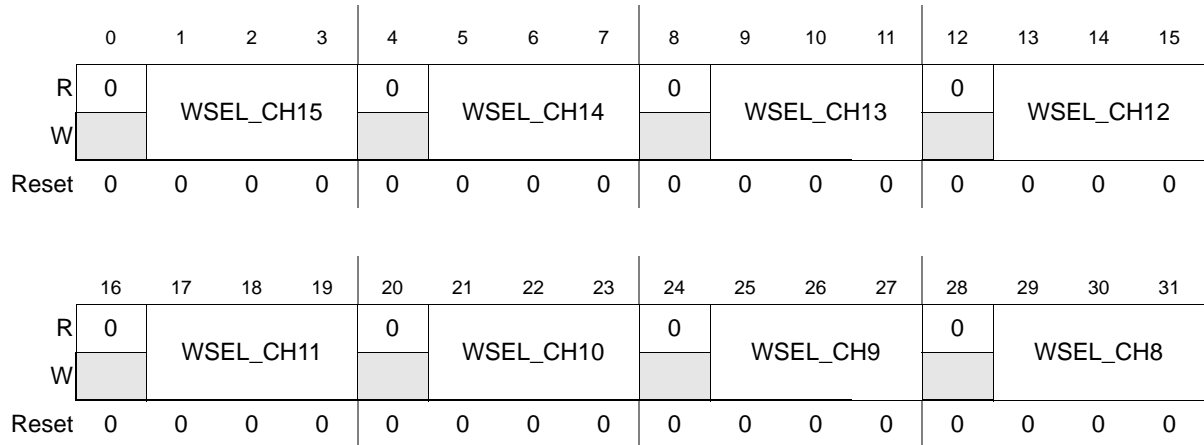


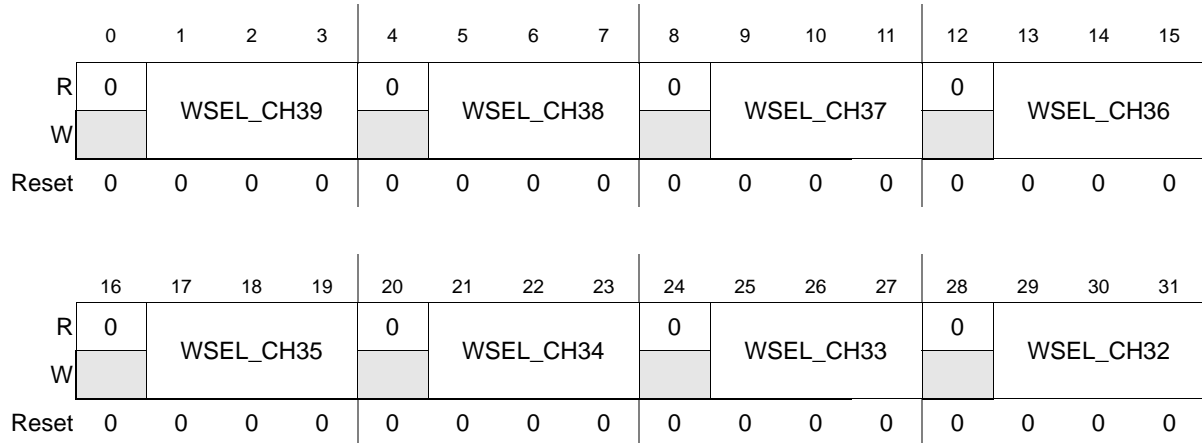
Figure 465. Channel Watchdog Select Register 1 (CWSELR1) – ADC\_0

**Table 415. CWSELR1 field descriptions – ADC\_0**

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

Offset: 0x02C0

Access: User read/write



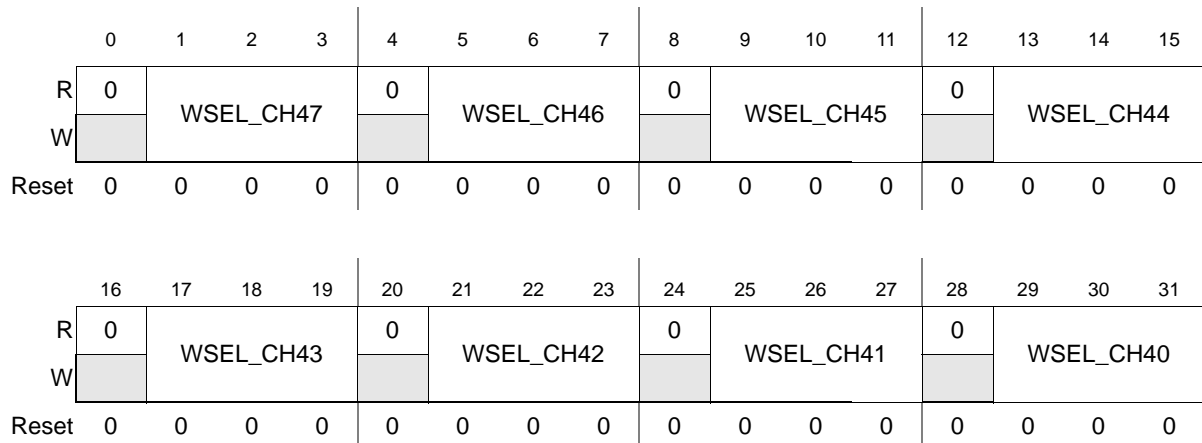
**Figure 466. Channel Watchdog Select Register 4 (CWSELR4) – ADC\_0**

**Table 416. CWSELR4 field descriptions – ADC\_0**

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

Offset: 0x02C4

Access: User read/write



**Figure 467. Channel Watchdog Select Register 5 (CWSELR5) – ADC\_0**

**Table 417. CWSELR5 field descriptions – ADC\_0**

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

Offset: 0x02C8

Access: User read/write

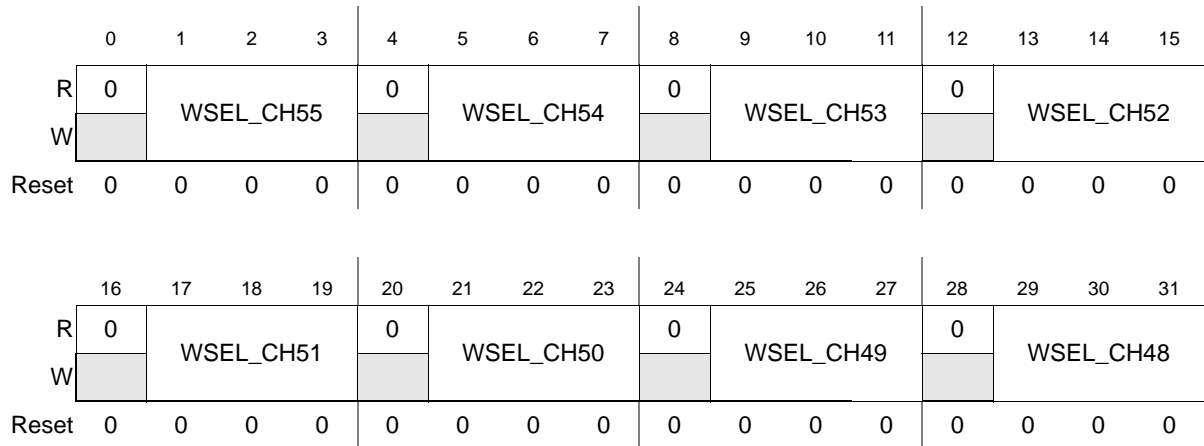


Figure 468. Channel Watchdog Select Register 6 (CWSELR6) – ADC\_0

Table 418. CWSELR6 field descriptions – ADC\_0

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

Offset: 0x02CC

Access: User read/write

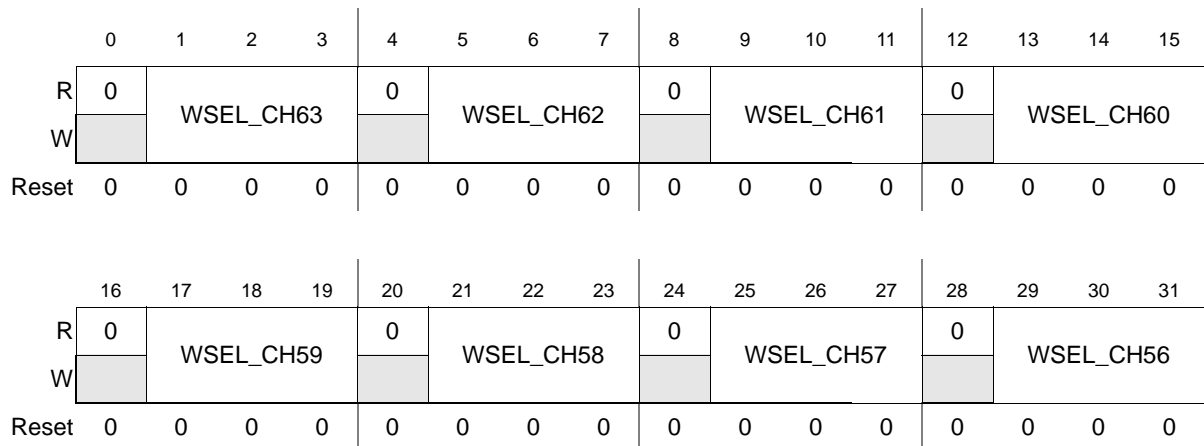


Figure 469. Channel Watchdog Select Register 7 (CWSELR7) – ADC\_0

Table 419. CWSELR7 field descriptions – ADC\_0

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

Offset: 0x02D0

Access: User read/write

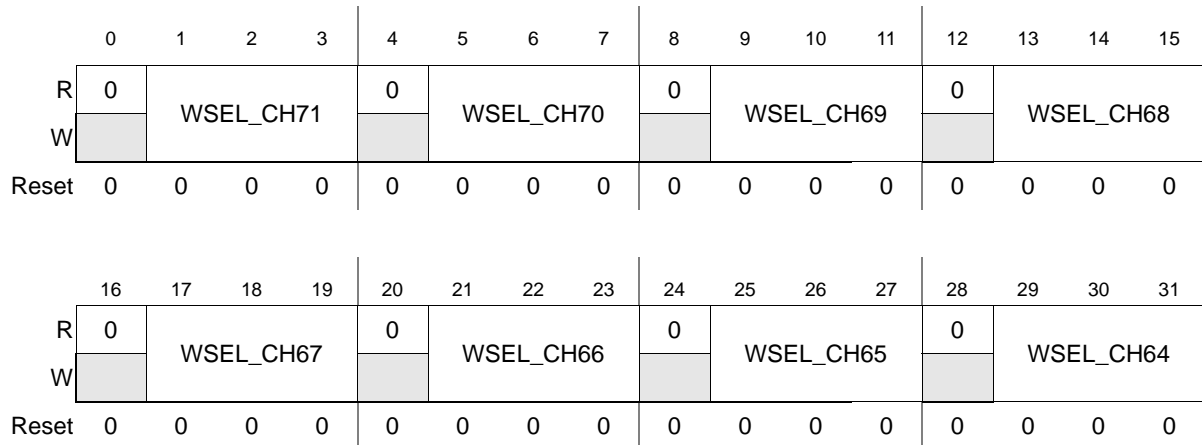


Figure 470. Channel Watchdog Select Register 8 (CWSELR8) – ADC\_0

Table 420. CWSELR8 field descriptions – ADC\_0

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

Offset: 0x02D4

Access: User read/write

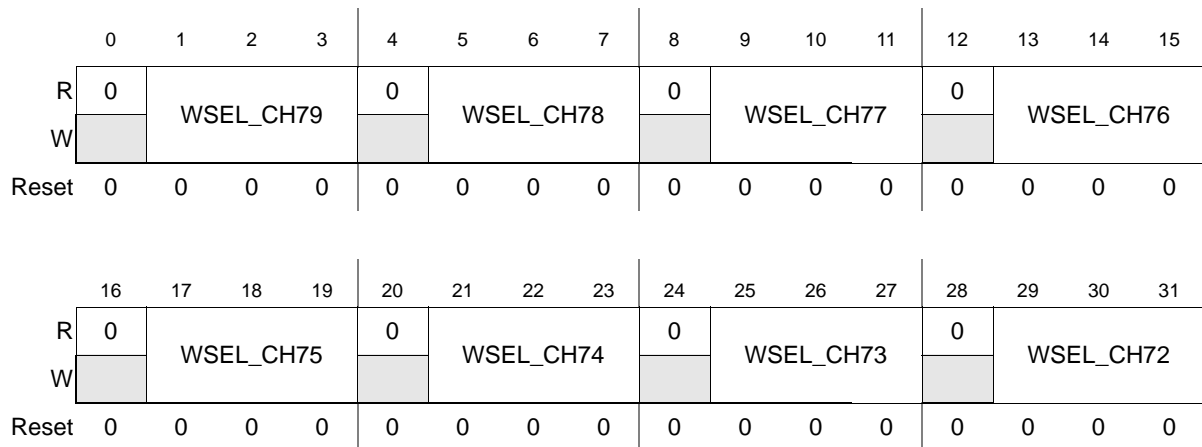


Figure 471. Channel Watchdog Select Register 9 (CWSELR9) – ADC\_0

Table 421. CWSELR9 field descriptions – ADC\_0

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .



Offset: 0x02D8

Access: User read/write

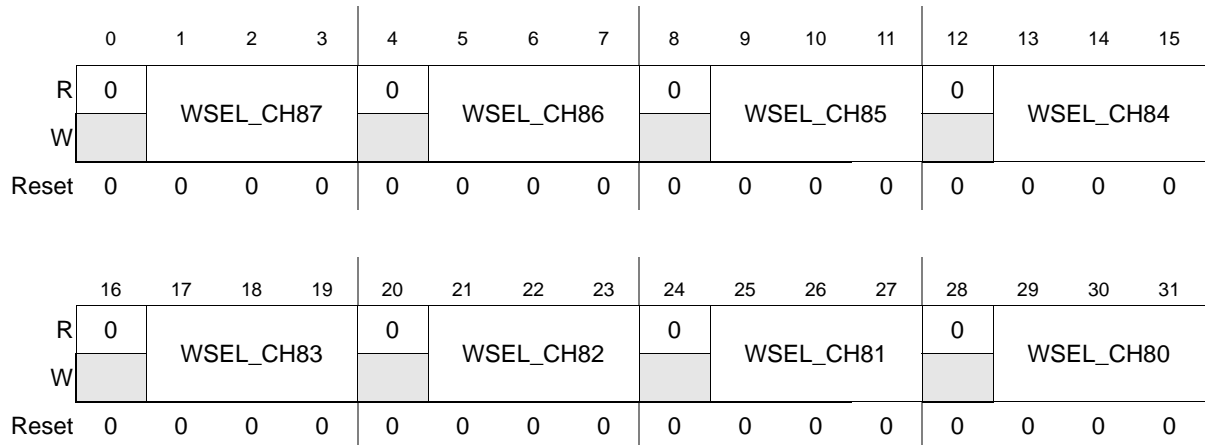


Figure 472. Channel Watchdog Select Register 10 (CWSELR10) – ADC\_0

Table 422. CWSELR10 field descriptions – ADC\_0

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

Offset: 0x02DC

Access: User read/write

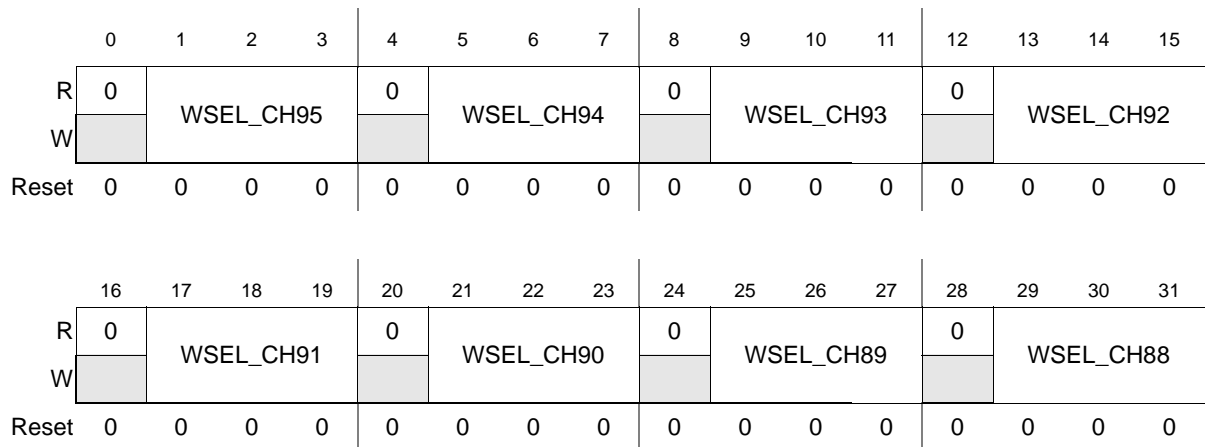


Figure 473. Channel Watchdog Select Register 11 (CWSELR11) – ADC\_0

Table 423. CWSELR11 field descriptions – ADC\_0

Field	Description
WSEL_CHn	See <a href="#">Table 414</a> .

ADC\_1

Offset: 0x02B0

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH7		0	0	WSEL_CH6		0	0	WSEL_CH5		0	0	WSEL_CH4	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	WSEL_CH3		0	0	WSEL_CH2		0	0	WSEL_CH1		0	0	WSEL_CH0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 474. Channel Watchdog Select Register 0 (CWSELR0) – ADC\_1

Table 424. CWSELR0 field descriptions – ADC\_1

Field	Description
WSEL_CHn	Channel Watchdog select for channel n 00: THRHLR0 register is selected 01: THRHLR1 register is selected 10: THRHLR2 register is selected 11: RESERVED

Offset: 0x02B4

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH1		0	0	WSEL_CH1		0	0	WSEL_CH1		0	0	WSEL_CH1	
W			5				4				3				2	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	WSEL_CH1		0	0	WSEL_CH1		0	0	WSEL_CH9		0	0	WSEL_CH8	
W			1				0									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 475. Channel Watchdog Select Register 1 (CWSELR1) – ADC\_1

Table 425. CWSELR1 field descriptions – ADC\_1

Field	Description
WSEL_CHn:	See <a href="#">Table 424</a> .

Offset: 0x02C0

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH3		0	0	WSEL_CH3		0	0	WSEL_CH3		0	0	WSEL_CH3	
W			9				8				7				6	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	WSEL_CH3		0	0	WSEL_CH3		0	0	WSEL_CH3		0	0	WSEL_CH3	
W			5				4				3				2	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 476. Channel Watchdog Select Register 4 (CWSELR4) – ADC\_1

Table 426. CWSELR4 field descriptions – ADC\_1

Field	Description
WSEL_CHn	See <a href="#">Table 424</a> .

Offset: 0x02C4

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WSEL_CH4		0	0	WSEL_CH4		0	0	WSEL_CH4		0	0	WSEL_CH4	
W			7				6				5				4	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	WSEL_CH4		0	0	WSEL_CH4		0	0	WSEL_CH4		0	0	WSEL_CH4	
W			3				2				1				0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 477. Channel Watchdog Select Register 5 (CWSELR5) – ADC\_1

Table 427. CWSELR5 field descriptions – ADC\_1

Field	Description
WSEL_CHn	See <a href="#">Table 424</a> .

28.4.11.2 Channel Watchdog Enable Register (CWENRx, x = [0...2])

[Table 428](#) shows the exact number of available channels.

Table 428. CWENR[0...2] register description

Register	Description	ADC
CWENR0	Watchdog enable bits for channel 0 to 15 (precision channels)	ADC_0
		ADC_1
CWENR1	Watchdog enable bits for channel 32 to 59 (standard channels)	ADC_0
	Watchdog enable bits for channel 32 to 39 (standard channels)	ADC_1
CWENR2	Watchdog enable bits for channel 64 to 95 (external multiplexed channels)	ADC_0

Address: Base + 0x02E0

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 478. Channel Watchdog Enable Register 0 (CWENR0)

Address: Base + 0x02E4

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN
W					59	58	57	56	55	54	53	52	51	50	49	48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN
W	47	46	43	44	43	42	41	40	39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 479. Channel Watchdog Enable Register 1 (CWENR1)

Address: Base + 0x02E4

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN
W									39	38	37	36	35	34	33	32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 480. Channel Watchdog Enable Register 1 (CWENR1) - ADC\_1

Address: Base + 0x02E08

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN
W	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN	CWEN
W	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 481. Channel Watchdog Enable Register 2 (CWENR2)

Table 429. CWENRx field descriptions

Field	Description
CWENn	Channel Watchdog enable When set (CWENn = 1) Watchdog feature is enabled for channel n.

28.4.11.3 Analog Watchdog Out of Range Register (AWORRx, x = [0..2])

Address: Base + 0x02F0

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AWOR_CH15	AWOR_CH14	AWOR_CH13	AWOR_CH12	AWOR_CH11	AWOR_CH10	AWOR_CH9	AWOR_CH8	AWOR_CH7	AWOR_CH6	AWOR_CH5	AWOR_CH4	AWOR_CH3	AWOR_CH2	AWOR_CH1	AWOR_CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 482. Analog Watchdog Out of Range Register 0 (AWORR0)

Address: Base + 0x02F4

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	AWOR_CH59	AWOR_CH58	AWOR_CH57	AWOR_CH56	AWOR_CH55	AWOR_CH54	AWOR_CH53	AWOR_CH52	AWOR_CH51	AWOR_CH50	AWOR_CH49	AWOR_CH48
W					w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AWOR_CH47	AWOR_CH46	AWOR_CH43	AWOR_CH44	AWOR_CH43	AWOR_CH42	AWOR_CH41	AWOR_CH40	AWOR_CH39	AWOR_CH38	AWOR_CH37	AWOR_CH36	AWOR_CH35	AWOR_CH34	AWOR_CH33	AWOR_CH32
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 483. Analog Watchdog Out of Range Register 1 (AWORR1) - ADC\_0



Address: Base + 0x02F4

Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	AWOR_CH39	AWOR_CH38	AWOR_CH37	AWOR_CH36	AWOR_CH35	AWOR_CH34	AWOR_CH33	AWOR_CH32
W									w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 484. Analog Watchdog Out of Range Register 1 (AWORR1) - ADC\_1

Address: Base + 0x02F8

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWOR_CH95	AWOR_CH94	AWOR_CH93	AWOR_CH92	AWOR_CH91	AWOR_CH90	AWOR_CH89	AWOR_CH88	AWOR_CH87	AWOR_CH86	AWOR_CH85	AWOR_CH84	AWOR_CH83	AWOR_CH82	AWOR_CH81	AWOR_CH80
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AWOR_CH79	AWOR_CH78	AWOR_CH77	AWOR_CH76	AWOR_CH75	AWOR_CH74	AWOR_CH73	AWOR_CH72	AWOR_CH71	AWOR_CH70	AWOR_CH69	AWOR_CH68	AWOR_CH67	AWOR_CH66	AWOR_CH65	AWOR_CH64
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 485. Analog Watchdog Out of Range Register 2 (AWORR2)

Table 430. AWORRx field descriptions

Field	Description
AWORR_CHn	When set indicates channel n converted data is out of range

## 29 Cross Triggering Unit (CTU)

### 29.1 Introduction

The Cross Triggering Unit (CTU) allows to synchronize an ADC conversion with a timer event from eMIOS (every mode which can generate a DMA request can trigger CTU) or PIT. To select which ADC channel must be converted on a particular timer event, the CTU provides the ADC with a 7-bit channel number. This channel number can be configured for each timer channel event by the application.

### 29.2 Main features

- Single cycle delayed trigger output. The trigger output is a combination of 64 (generic value) input flags/events connected to different timers in the system.
- One event configuration register dedicated to each timer event allows to define the corresponding ADC channel.
- Acknowledgment signal to eMIOS/PIT for clearing the flag
- Synchronization with ADC to avoid collision

### 29.3 Block diagram

The CTU block diagram is shown in [Figure 486](#).

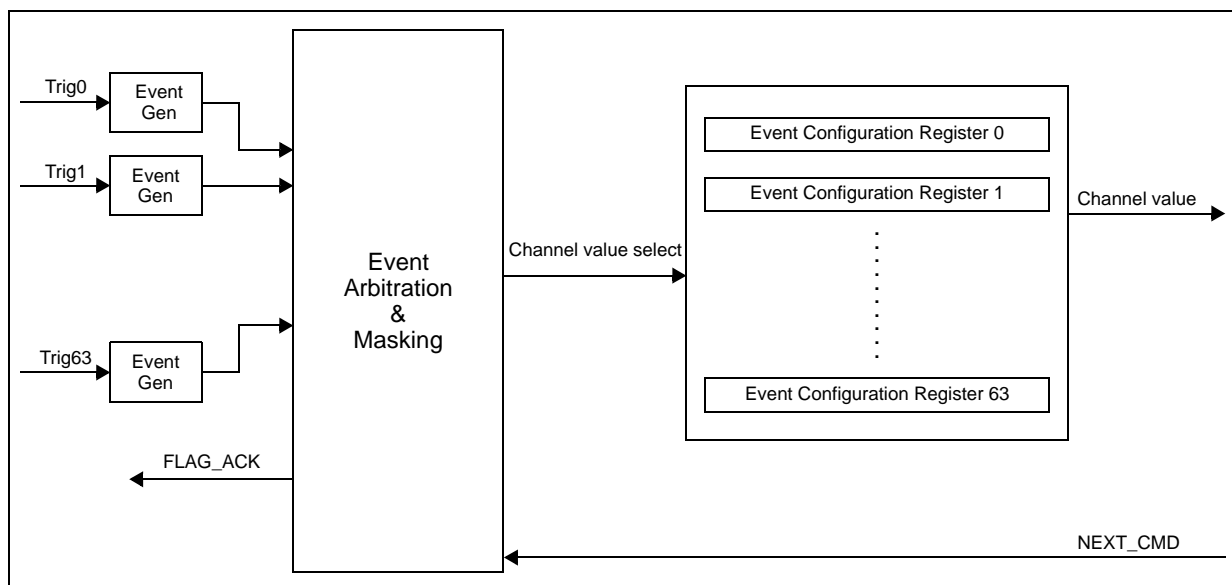


Figure 486. Cross Triggering Unit block diagram



## 29.4 Memory map and register descriptions

The CTU registers are listed in [Table 431](#). Every register can have 32-bit access. The base address of the CTU is 0xFFE6\_4000.

**Table 431. CTU memory map**

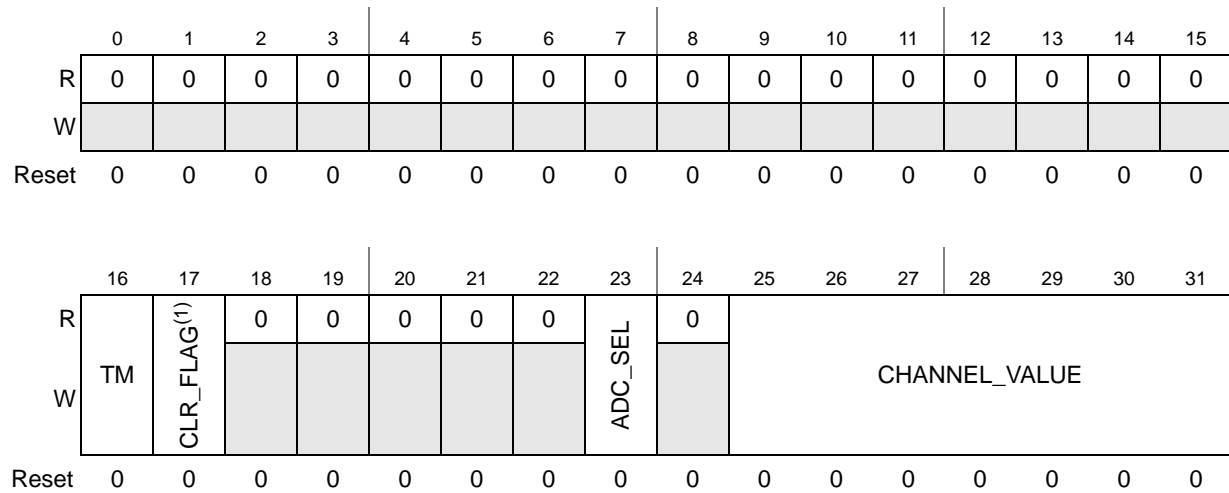
Base address: 0xFFE6_4000		
Address offset	Register	Location
0x000–0x02F	Reserved	
0x030–0x12C	<a href="#">Event Configuration Registers 0...63</a> (CTU_EVTCFGR0...63)	<a href="#">on page 817</a>

### 29.4.1 Event Configuration Registers (CTU\_EVTCFGRx) (x = 0...63)

**Figure 487. Event Configuration Registers (CTU\_EVTCFGRx) (x = 0...63)**

Offsets: 0x030–0x12C

Access: Read/write



1. This bit implementation is generic based and implemented only for inputs mapped to PIT event flags.

**Table 432. CTU\_EVTCFGRx field descriptions**

Field	Description
TM	Trigger Mask 0: Trigger masked 1: Trigger enabled
CLR_FLAG	To provide flag_ack through software 1: Flag_ack is forced to '1' for the particular event 0: Flag_ack is dependent on flag servicing
ADC_SEL	This bit selects the ADC number. 0: 10-bit ADC0 is selected 1: 12-bit ADC1 is selected
CHANNEL_VALUE	These bits provide the ADC channel number to be converted. Valid values are 0b0 to 0b1011111 (decimal 95).

These registers contain the ADC channel number to be converted when the timer event occurs. The CLR\_FLAG is used to clear the respective timer event flag by software (this applies only to the PIT as the eMIOS flags are automatically cleared by the CTU).

The CLR\_FLAG bit has to be used cautiously as setting this bit may result in a loss of events.

The event input can be masked by writing '0' to bit TM of the CTU\_EVTCFGR register. Writing '1' to bit TM enables the CTU triggering and automatically disables the DMA connection for the corresponding eMIOS channel.

*Note: The CTU tracks issued conversion requests to the ADC. When the ADC is being triggered by the CTU and there is a need to shut down the ADC, the ADC must be allowed to complete conversions before being shutdown. This ensures that the CTU is notified of completion; if the ADC is shut down while performing a CTU triggered conversion, the CTU is not notified and will not be able to trigger further conversions until the device is reset.*

## 29.5 Functional description

This peripheral is used to synchronize ADC conversions with timer events (from eMIOS or PIT). When a timer event occurs, the CTU triggers an ADC conversion providing the ADC channel number to be converted. In case concurrent events occur the priority is managed according to the index of the timer event. The trigger output is a single cycle pulse used to trigger ADC conversion of the channel number provided by the CTU.

Each trigger input from the CTU is connected to the Event Trigger signal of an eMIOS channel. The assignment between eMIOS outputs and CTU trigger inputs is defined in [Table 433](#).

**Table 433. Trigger source**

CTU trigger No.	Module	Source
0	eMIOS 0	Channel_0
1	eMIOS 0	Channel_1
2	eMIOS 0	Channel_2
3	eMIOS 0	Channel_3
4	eMIOS 0	Channel_4
5	eMIOS 0	Channel_5
6	eMIOS 0	Channel_6
7	eMIOS 0	Channel_7
8	eMIOS 0	Channel_8
9	eMIOS 0	Channel_9
10	eMIOS 0	Channel_10
11	eMIOS 0	Channel_11
12	eMIOS 0	Channel_12
13	eMIOS 0	Channel_13
14	eMIOS 0	Channel_14

Table 433. Trigger source (continued)

CTU trigger No.	Module	Source
15	eMIOS 0	Channel_15
16	eMIOS 0	Channel_16
17	eMIOS 0	Channel_17
18	eMIOS 0	Channel_18
19	eMIOS 0	Channel_19
20	eMIOS 0	Channel_20
21	eMIOS 0	Channel_21
22	eMIOS 0	Channel_22
23	PIT	PIT_3
24	eMIOS 0	Channel_24
25	eMIOS 0	Channel_25
26	eMIOS 0	Channel_26
27	eMIOS 0	Channel_27
28	eMIOS 0	Channel_28
29	eMIOS 0	Channel_29
30	eMIOS 0	Channel_30
31	eMIOS 0	Channel_31
32	eMIOS 1	Channel_0
33	eMIOS 1	Channel_1
34	eMIOS 1	Channel_2
35	eMIOS 1	Channel_3
36	eMIOS 1	Channel_4
37	eMIOS 1	Channel_5
38	eMIOS 1	Channel_6
39	eMIOS 1	Channel_7
40	eMIOS 1	Channel_8
41	eMIOS 1	Channel_9
42	eMIOS 1	Channel_10
43	eMIOS 1	Channel_11
44	eMIOS 1	Channel_12
45	eMIOS 1	Channel_13
46	eMIOS 1	Channel_14
47	eMIOS 1	Channel_15
48	eMIOS 1	Channel_16
49	eMIOS 1	Channel_17

**Table 433. Trigger source (continued)**

CTU trigger No.	Module	Source
50	eMIOS 1	Channel_18
51	eMIOS 1	Channel_19
52	eMIOS 1	Channel_20
53	eMIOS 1	Channel_21
54	eMIOS 1	Channel_22
55	PIT	PIT_7
56	eMIOS 1	Channel_24
57	eMIOS 1	Channel_25
58	eMIOS 1	Channel_26
59	eMIOS 1	Channel_27
60	eMIOS 1	Channel_28
61	eMIOS 1	Channel_29
62	eMIOS 1	Channel_30
63	eMIOS 1	Channel_31

Each event has a dedicated configuration register (CTU\_EVTCFGR). These registers store a channel number which is used to communicate which channel needs to be converted.

In case several events are pending for ADC request, the priority is managed according to the timer event index. The lowest index has the highest priority. Once an event has been serviced (conversion requested to ADC) the eMIOS flag is cleared by the CTU and next prior event is handled.

The acknowledgment signal can be forced to '1' by setting the CLR\_FLAG bit of the CTU\_EVTCFGR register. These bits are implemented for only those input flags to which PIT flags are connected. Providing these bits offers the option of clearing PIT flags by software.

### 29.5.1 Channel value

The channel value stored in an event configuration register is demultiplexed to 7 bits and then provided to the ADC.

The mapping of the channel number value to the corresponding ADC channel is provided in [Table 433](#).

**Table 434. CTU-to-ADC channel assignment**

10-bit ADC			12-bit ADC		
10-bit ADC_0 signal name	10-bit ADC_0 channel #	Channel # in CTU_EVTCFGRx	12-bit ADC_1 signal name	12-bit ADC_1 channel #	Channel # in CTU_EVTCFGRx
ADC0_P[0]	CH0	0	ADC1_P[0]	CH0	0
ADC0_P[1]	CH1	1	ADC1_P[1]	CH1	1

Table 434. CTU-to-ADC channel assignment (continued)

10-bit ADC			12-bit ADC		
10-bit ADC_0 signal name	10-bit ADC_0 channel #	Channel # in CTU_EVTFCGRx	12-bit ADC_1 signal name	12-bit ADC_1 channel #	Channel # in CTU_EVTFCGRx
ADC0_P[2]	CH2	2	ADC1_P[2]	CH2	2
ADC0_P[3]	CH3	3	ADC1_P[3]	CH3	3
ADC0_P[4]	CH4	4	ADC1_P[4]	CH4	4
ADC0_P[5]	CH5	5	ADC1_P[5]	CH5	5
ADC0_P[6]	CH6	6	ADC1_P[6]	CH6	6
ADC0_P[7]	CH7	7	ADC1_P[7]	CH7	7
ADC0_P[8]	CH8	8	ADC1_P[8]	CH8	8
ADC0_P[9]	CH9	9	ADC1_P[9]	CH9	9
ADC0_P[10]	CH10	10	ADC1_P[10]	CH10	10
ADC0_P[11]	CH11	11	ADC1_P[11]	CH11	11
ADC0_P[12]	CH12	12	ADC1_P[12]	CH12	12
ADC0_P[13]	CH13	13	ADC1_P[13]	CH13	13
ADC0_P[14]	CH14	14	ADC1_P[14]	CH14	14
ADC0_P[15]	CH15	15	ADC1_P[15]	CH15	15
ADC0_S[0]	CH32	32	ADC1_S[0]	CH32	32
ADC0_S[1]	CH33	33	ADC1_S[1]	CH33	33
ADC0_S[2]	CH34	34	ADC1_S[2]	CH34	34
ADC0_S[3]	CH35	35	ADC1_S[3]	CH35	35
ADC0_S[4]	CH36	36	ADC1_S[4]	CH36	36
ADC0_S[5]	CH37	37	ADC1_S[5]	CH37	37
ADC0_S[6]	CH38	38	ADC1_S[6]	CH38	38
ADC0_S[7]	CH39	39	ADC1_S[7]	CH39	39
ADC0_S[8]	CH40	40			
ADC0_S[9]	CH41	41			
ADC0_S[10]	CH42	42			
ADC0_S[11]	CH43	43			
ADC0_S[12]	CH44	44			
ADC0_S[13]	CH45	45			
ADC0_S[14]	CH46	46			
ADC0_S[15]	CH47	47			
ADC0_S[16]	CH48	48			
ADC0_S[17]	CH49	49			

Table 434. CTU-to-ADC channel assignment (continued)

10-bit ADC			12-bit ADC		
10-bit ADC_0 signal name	10-bit ADC_0 channel #	Channel # in CTU_EVTFCGRx	12-bit ADC_1 signal name	12-bit ADC_1 channel #	Channel # in CTU_EVTFCGRx
ADC0_S[18]	CH50	50			
ADC0_S[19]	CH51	51			
ADC0_S[20]	CH52	52			
ADC0_S[21]	CH53	53			
ADC0_S[22]	CH54	54			
ADC0_S[23]	CH55	55			
ADC0_S[24]	CH56	56			
ADC0_S[25]	CH57	57			
ADC0_S[26]	CH58	58			
ADC0_S[27]	CH59	59			
ADC0_X[0]	CH64: CH71	64:71			
ADC0_X[1]	CH72: CH79	72:79			
ADC0_X[2]	CH80: CH87	80:87			
ADC0_X[3]	CH88: CH95	88:95			

CTU channel mapping should be taken into consideration when programming an event configuration register. For example, if the channel value of any event configuration register is programmed to 16, it will actually correspond to ADC channel 32 and conversion will occur for this channel.

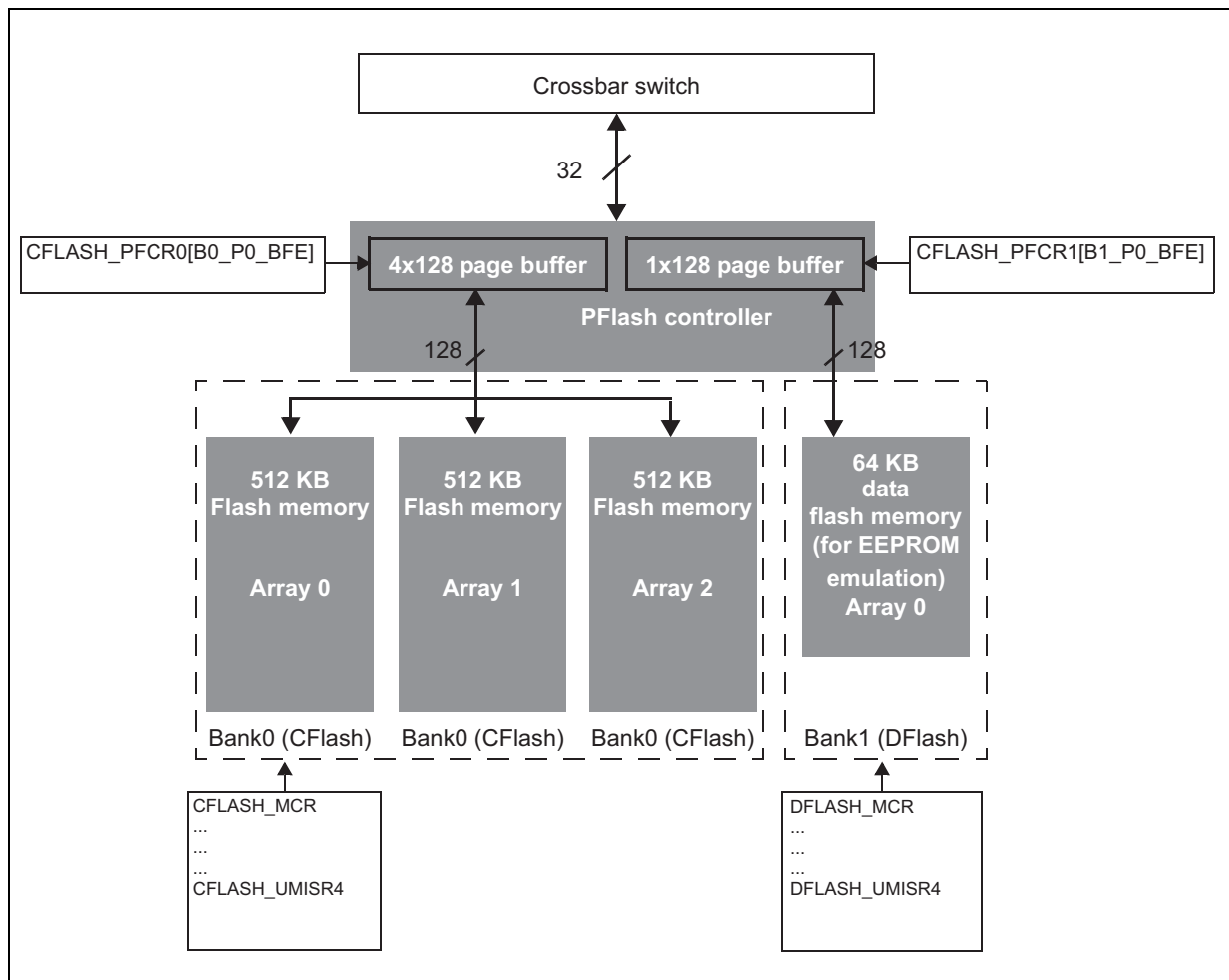
## 30 Flash Memory

### 30.1 Introduction

The flash memory comprises a platform flash memory controller (PFlash) interface and the following flash memory arrays:

- Three arrays of 512 KB for code (CFlash)
- One array of 64 KB for data (DFlash)

The flash memory architecture of this device is illustrated in *Figure 488*.



**Figure 488. Flash memory architecture**

The primary function of the flash memory module is to serve as electrically programmable and erasable nonvolatile memory.

Nonvolatile memory may be used for instruction and/or data storage.

The module is a nonvolatile solid-state silicon memory device consisting of:

- Blocks (also called “sectors”) of single transistor storage elements
- An electrical means for selectively adding (programming) and removing (erasing) charge from these elements
- A means of selectively sensing (reading) the charge stored in these elements

The flash memory module is arranged as two functional units:

- The flash memory core
- The memory interface

The flash memory core is composed of arrayed nonvolatile storage elements, sense amplifiers, row decoders, column decoders and charge pumps. The arrayed storage elements in the flash memory core are subdivided into physically separate units referred to as blocks (or sectors).

The memory interface contains the registers and logic which control the operation of the flash memory core. The memory interface is also the interface between the flash memory module and a platform flash memory controller. It contains the ECC logic and redundancy logic.

A platform flash memory controller connects the flash memory module to a system bus, and contains all system level customization required for the device application.

## 30.2 Main features

**Table 435. Flash memory features**

Feature	CFlash	DFlash
High read parallelism (128 bits)	Yes	
Error Correction Code (SEC-DED) to enhance data retention	Yes	
Double Word Program (64 bits)	Yes	
Sector erase	Yes	
Single bank—Read-While-Write (RWW)	No	
Erase Suspend	Yes	
Program Suspend	No	
Software programmable program/erase protection to avoid unwanted writings	Yes	
Censored Mode against piracy	Yes	
Shadow Sector available	Yes	No
One-Time Programmable (OTP) area in Test Flash block	Yes	
Boot sectors	Yes	No

## 30.3 Block diagram

The flash memory module contains one Matrix Module, composed of a single bank (Bank 0) normally used for code storage. RWW operations are not possible.



Modify operations are managed by an embedded Flash Memory Program/Erase Controller (FPEC). Commands to the FPEC are given through a User Registers Interface.

The read data bus is 128 bits wide, while the flash memory registers are on a separate bus 32 bits wide addressed in the user memory map.

The high voltages needed for program/erase operations are generated internally.

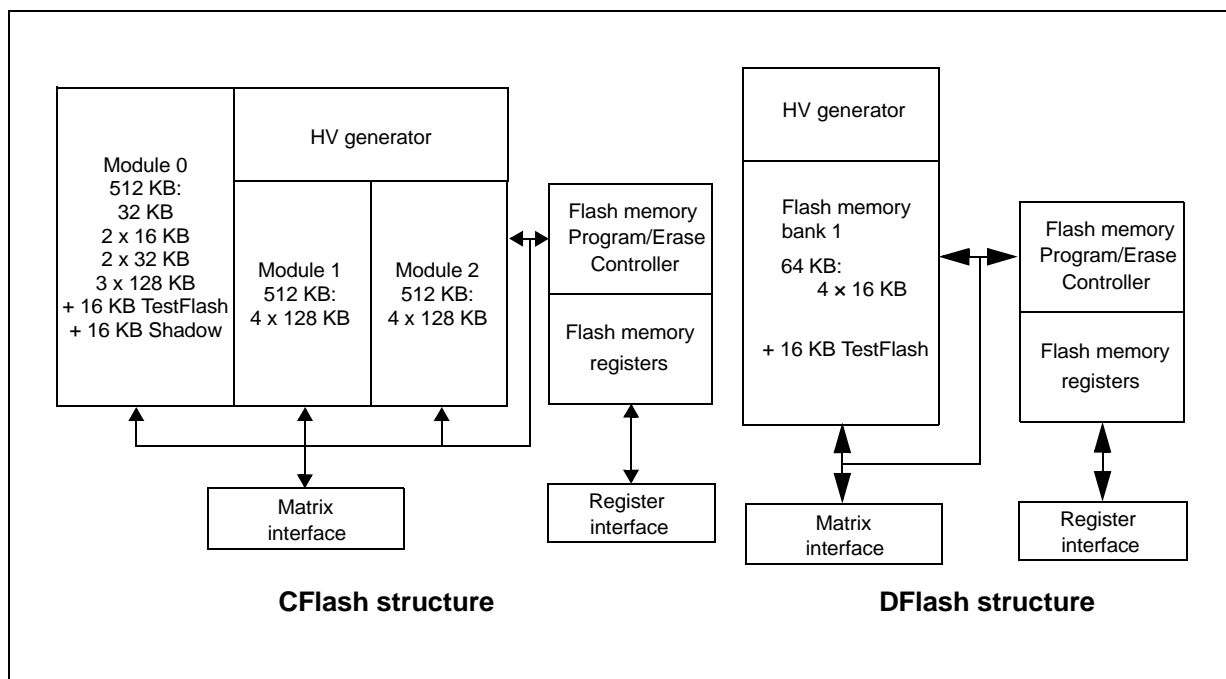


Figure 489. CFlash and DFlash module structures

## 30.4 Functional description

### 30.4.1 Module structure

The flash memory module is addressable by Double Word (64 bits) for program, and page (128 bits) for read. Reads to the flash memory always return 128 bits, although read page buffering may be done in the platform flash memory controller.

Each read of the flash memory module retrieves a page, or four consecutive words (128 bits) of information. The address for each word retrieved within a page differs from the other addresses in the page only by address bits (3:2).

The flash memory module supports fault tolerance through Error Correction Code (ECC) or error detection, or both. The ECC implemented within the flash memory module will correct single bit failures and detect double bit failures.

The flash memory module uses an embedded hardware algorithm implemented in the Memory Interface to program and erase the flash memory core.

The embedded hardware algorithm includes control logic that works with software block enables and software lock mechanisms to guard against accidental program/erase.

The hardware algorithm performs the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to guarantee data integrity and reliability.

In the flash memory module, logic levels are defined as follows:

- A programmed bit reads as logic level 0 (or low).
- An erased bit reads as logic level 1 (or high).

Program and erase of the flash memory module requires multiple system clock cycles to complete.

The erase sequence may be suspended.

The program and erase sequences may be aborted.

## 30.4.2 Flash memory module sectorization

### 30.4.2.1 CFlash module sectorization

The CFlash module supports 1.5 MBoF of user memory, plus 16 KB of test memory (a portion of which is One-Time Programmable by the user). An extra 16 KB sector is available as Shadow space usable for user option bits and censorship settings.

The module is composed of a single bank (Bank 0): Read-While-Write is not supported.

Bank 0 of the module is divided in 18 sectors including a reserved sector, named TestFlash, in which some One-Time Programmable (OTP) user data are stored, as well as a Shadow Sector in which user erasable configuration values can be stored.

The matrix module sectorization is shown in [Table 436](#).

**Table 436. CFlash module sectorization**

CFlash			Sector address range	Sector size	Address space	Address space locking register	
Bank	Sector	Module				CFLASH_LML	CFLASH_HBL
0	0	0	0x00000000–0x00007FFF	32 KB	Low	LLK0	
0	1	0	0x00008000–0x0000BFFF	16 KB	Low	LLK1	
0	2	0	0x0000C000–0x0000FFFF	16 KB	Low	LLK2	
0	3	0	0x00010000–0x00017FFF	32 KB	Low	LLK3	
0	4	0	0x00018000–0x0001FFFF	32 KB	Low	LLK4	
0	5	0	0x00020000–0x0003FFFF	128 KB	Low	LLK5	
0	6	0	0x00040000–0x0005FFFF	128 KB	Mid	MLK0	
0	7	0	0x00060000–0x0007FFFF	128 KB	Mid	MLK1	
0	8	1	0x00080000–0x0009FFFF	128 KB	High		HLK0
0	9	1	0x000A0000–0x000BFFFF	128 KB	High		HLK1
0	10	1	0x000C0000–0x000DFFFF	128 KB	High		HLK2
0	11	1	0x000E0000–0x000FFFFF	128 KB	High		HLK3
0	12	2	0x00100000–0x0011FFFF	128 KB	High		HLK4
0	13	2	0x00120000–0x0013FFFF	128 KB	High		HLK5

**Table 436. CFlash module sectorization (continued)**

CFlash			Sector address range	Sector size	Address space	Address space locking register	
Bank	Sector	Module				CFLASH_LML	CFLASH_HBL
0	14	2	0x00140000–0x0015FFFF	128 KB	High		HLK6
0	15	2	0x00160000–0x0017FFFF	128 KB	High		HLK7
0	Shadow	0	0x00200000–0x00203FFF	16 KB	Shadow	TSLK	
0	Test	0	0x00400000–0x00403FFF	16 KB	Test	TSLK	

The division into blocks of the flash memory module is also used to implement independent erase/program protection. A software mechanism is provided to independently lock/unlock each block in low and mid address space against program and erase.

### 30.4.2.2 DFlash module sectorization

The DFlash module supports 64 KB of user memory, plus 16 KB of test memory (a portion of which is One-Time Programmable by the user).

The module is composed of a single bank (Bank 0): Read-While-Write is not supported.

Bank 0 of the 80 KB module is divided in four sectors. Bank 0 also contains a reserved sector named TestFlash in which some One-Time Programmable user data are stored.

The sectorization of the 80 KB matrix module is shown in [Table 437](#).

**Table 437. DFlash module sectorization**

Bank	Sector	Addresses	Size (KB)	Address space	DFLASH_LML field for locking the address space
0	0	0x00800000–0x00803FFF	16	Low	LLK0
	1	0x00804000–0x00807FFF			LLK1
	2	0x00808000–0x0080BFFF			LLK2
	3	0x0080C000–0x0080FFFF			LLK3
	Test	0x00C00000–0x00C03FFF		Test	TSLK

The flash memory module is divided into blocks also to implement independent erase/program protection. A software mechanism is provided to independently lock/unlock each block in low and mid address space against program and erase.

### 30.4.3 TestFlash block

A TestFlash block is available in both the CFlash and DFlash modules. The TestFlash block exists outside the normal address space and is programmed and read independently of the other blocks. The independent TestFlash block is included to also support systems which require nonvolatile memory for security or to store system initialization information, or both.

A section of the TestFlash is reserved to store the nonvolatile information related to Redundancy, Configuration and Protection.

The ECC is also applied to TestFlash.

The structure of the TestFlash sector is detailed in [Table 438](#) and [Table 439](#).

**Table 438. CFlash TestFlash structure**

Name	Description	Addresses	Size
—	User OTP area	0x400000–0x401FFF	8192 bytes
—	Reserved	0x402000–0x403CFF	7424 bytes
—	User OTP area	0x403D00–0x403DE7	232 bytes
CFLASH_NVLML	CFlash Nonvolatile Low/Mid Address Space Block Locking Register	0x403DE8–0x403DEF	8 bytes
CFLASH_NVHBL	CFlash Nonvolatile High Address Space Block Locking Register	0x403DF0–0x403DF7	8 bytes
CFLASH_NVSL	CFlash Nonvolatile Secondary Low/mid Address Space Block Locking Register	0x403DF8–0x403DFF	8 bytes
—	User OTP area	0x403E00–0x403EFF	256 bytes
—	Reserved	0x403F00–0x403FFF	256 bytes

**Table 439. DFlash TestFlash structure**

Name	Description	Addresses	Size
—	User OTP area	0xC00000–0xC01FFF	8192 bytes
—	Reserved	0xC02000–0xC03CFF	7424 bytes
—	User OTP area	0xC03D00–0xC03DE7	232 bytes
DFLASH_NVLML	DFlash Nonvolatile Low/Mid Address Space Block Locking Register	0xC03DE8–0xC03DEF	8 bytes
—	Reserved	0xC03DF0–0xC03DF7	8 bytes
DFLASH_NVSL	DFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register	0xC03DF8–0xC03DFF	8 bytes
—	User OTP area	0xC03E00–0xC03EFF	256 bytes
—	Reserved	0xC03F00–0xC03FFF	256 bytes

Erase of the TestFlash block is always locked.

Programming of the TestFlash block has similar restrictions as the array in terms of how ECC is calculated. Only one programming operation is allowed per 64-bit ECC segment.

The first 8 KB of TestFlash block may be used for user defined functions (possibly to store serial numbers, other configuration words or factory process codes). Locations of the TestFlash other than the first 8 KB of OTP area cannot be programmed by the user application.

#### 30.4.4 Shadow sector

The shadow sector is only present in the CFlash module.

User Mode program and erase of the shadow sector are enabled only when CFLASH\_MCR[PEAS] is high.

The shadow sector may be locked/unlocked against program or erase by using the CFLASH\_LML[TSLK] and CFLASH\_SLL[STSLK] fields.

Programming of the shadow sector has similar restrictions as the array in terms of how ECC is calculated. Only one programming operation is allowed per 64-bit ECC segment between erases.

Erase of the shadow sector is done similarly to a sector erase.

The shadow sector contains specified data that are needed for user features.

The user area of shadow sector may be used for user defined functions (possibly to store boot code, other configuration words or factory process codes).

The structure of the shadow sector is detailed in [Table 440](#).

**Table 440. Shadow sector structure**

Name	Description	Addresses	Size (bytes)
—	User area	0x200000–0x203DCF	15824
—	Reserved	0x203DD0–0x203DD7	8
NVPWD0–1	Nonvolatile Private Censorship PassWord 0–1 registers	0x203DD8–0x203DDF	8
NVSCC0–1	Nonvolatile System Censorship Control 0–1 registers	0x203DE0–0x203DE7	8
—	Reserved	0x203DE8–0x203DFF	24
NVPFAPR	Nonvolatile Platform Flash Memory Access Protection Register	0x203E00–0x203E07	8
—	Reserved	0x203E08–0x203E17	16
NVUSRO	Nonvolatile User Options register	0x203E18–0x203E1F	8
—	Reserved	0x203E20–0x203FFF	480

### 30.4.5 User mode operation

In User Mode the flash memory module may be read and written (register writes and interlock writes), programmed or erased.

The default state of the flash memory module is read.

The main, shadow and test address space can be read only in the read state.

The majority of CFlash and DFlash memory-mapped registers can be read even when the CFlash or DFlash is in power-down or low-power mode. The exceptions are as follows:

- CFlash
  - UT0[MRE, MRV, AIS, DSI0:7]
  - UT1
  - UT2
- DFlash
  - UT0[MRE, MRV, AIS, DSI0:7]

- UT1
- UT2

The flash memory module enters the read state on reset.

The module is in the read state under two sets of conditions:

- The read state is active when the module is enabled (User Mode Read).
- The read state is active when the ERS and ESUS fields in the corresponding MCR (CFLASH\_MCR or DFLASH\_MCR) are 1 and the PGM field is 0 (Erase Suspend).

Flash memory core reads return 128 bits (1 Page = 2 Double Words).

Registers reads return 32 bits (1 Word).

Flash memory core reads are done through the platform flash memory controller.

Registers reads to unmapped register address space will return all 0's.

Registers writes to unmapped register address space will have no effect.

Attempted array reads to invalid locations will result in indeterminate data. Invalid locations occur when blocks that do not exist in non  $2^n$  array sizes are addressed.

Attempted interlock writes to invalid locations will result in an interlock occurring, but attempts to program these blocks will not occur since they are forced to be locked. Erase will occur to selected and unlocked blocks even if the interlock write is to an invalid location.

Simultaneous Read cycle on the Flash Matrix and Read/Write cycles on the registers are possible. On the contrary, registers read/write accesses simultaneous to a Flash Matrix interlock write are forbidden.

### 30.4.6 Reset

A reset is the highest priority operation for the flash memory module and terminates all other operations.

The flash memory module uses reset to initialize register and status bits to their default reset values. If the flash memory module is executing a Program or Erase operation (PGM = 1 or ERS = 1 in CFLASH\_MCR or DFLASH\_MCR) and a reset is issued, the operation will be suddenly terminated and the module will disable the high voltage logic without damage to the high voltage circuits. Reset terminates all operations and forces the flash memory module into User Mode ready to receive accesses. Reset and power-off must not be used as a systematic way to terminate a Program or Erase operation.

After reset is negated, read register access may be done, although it should be noted that registers that require updating from shadow information, or other inputs, may not read updated values until the DONE field (in CFLASH\_MCR or DFLASH\_MCR) transitions. The DONE field may be polled to determine if the flash memory module has transitioned out of reset. Notice that the registers cannot be written until the DONE field is high.

### 30.4.7 Power-down mode

All flash memory DC current sources can be turned off in power-down mode, so that all power dissipation is due only to leakage in this mode. Flash memory power-down mode can be selected at ME\_<mode>\_MC.

Reads from or writes to the module are not possible in power-down mode.

When enabled the flash memory module returns to its pre-disable state in all cases unless in the process of executing an erase high voltage operation at the time of disable.

If the flash memory module is disabled during an erase operation, MCR[ESUS] bit is programmed to '1'. The user may resume the erase operation at the time the module is enabled by programming MCR[ESUS] = 0. MCR[EHV] must be high to resume the erase operation.

If the flash memory module is disabled during a program operation, the operation will in any case be completed and the power-down mode will be entered only after the programming ends.

The user should realize that, if the flash memory module is put in power-down mode and the interrupt vectors remain mapped in the flash memory address space, the flash memory module will greatly increase the interrupt response time by adding several wait-states.

It is forbidden to enter in low power mode when the power-down mode is active.

### 30.4.8 Low power mode

The low power mode turns off most of the DC current sources within the flash memory module. Flash memory low power mode can be selected at ME\_<mode>\_MC.

The module (flash memory core and registers) is not accessible for read or write once it enters low power mode.

Wake-up time from low power mode is faster than wake-up time from power-down mode.

When exiting from low power mode the flash memory module returns to its pre-sleep state in all cases unless it is executing an erase high voltage operation at the time low power mode is entered.

If the flash memory module enters low power mode during an erase operation, MCR[ESUS] is programmed to '1'. The user may resume the erase operation at the time the module exits low power mode by programming MCR[ESUS] = 0. MCR[EHV] must be high to resume the erase operation.

If the flash memory module enters low power mode during a program operation, the operation will be in any case completed and the low power mode will be entered only after the programming end.

It is forbidden to enter power-down mode when the low power mode is active.

## 30.5 Register description

The CFlash and DFlash modules have respective sets of memory mapped registers. The CFlash register mapping is shown in [Table 441](#). The DFlash register mapping is shown in [Table 442](#).

**Table 441. CFlash registers**

Address offset	Register	Location
0x0000	CFlash Module Configuration Register (CFLASH_MCR)	<a href="#">on page 833</a>
0x0004	CFlash Low/Mid Address Space Block Locking Register (CFLASH_LML)	<a href="#">on page 839</a>

**Table 441. CFlash registers (continued)**

Address offset	Register	Location
0x0008	CFlash High Address Space Block Locking Register (CFLASH_HBL)	<a href="#">on page 843</a>
0x000C	CFlash Secondary Low/Mid Address Space Block Locking Register (CFLASH_SLL)	<a href="#">on page 846</a>
0x0010	CFlash Low/Mid Address Space Block Select Register (CFLASH_LMS)	<a href="#">on page 853</a>
0x0014	CFlash High Address Space Block Select Register (CFLASH_HBS)	<a href="#">on page 854</a>
0x0018	CFlash Address Register (CFLASH_ADR)	<a href="#">on page 856</a>
0x0028–0x0038	Reserved	
0x003C	CFlash User Test 0 register (CFLASH_UT0)	<a href="#">on page 857</a>
0x0040	CFlash User Test 1 register (CFLASH_UT1)	<a href="#">on page 859</a>
0x0044	CFlash User Test 2 register (CFLASH_UT2)	<a href="#">on page 859</a>
0x0048	CFlash User Multiple Input Signature Register 0 (CFLASH_UMISR0)	<a href="#">on page 860</a>
0x004C	CFlash User Multiple Input Signature Register 1 (CFLASH_UMISR1)	<a href="#">on page 861</a>
0x0050	CFlash User Multiple Input Signature Register 2 (CFLASH_UMISR2)	<a href="#">on page 862</a>
0x0054	CFlash User Multiple Input Signature Register 3 (CFLASH_UMISR3)	<a href="#">on page 863</a>
0x0058	CFlash User Multiple Input Signature Register 4 (CFLASH_UMISR4)	<a href="#">on page 864</a>

**Table 442. DFlash registers**

Address offset	Register name	Location
0x0000	DFlash Module Configuration Register (DFLASH_MCR)	<a href="#">on page 869</a>
0x0004	DFlash Low/Mid Address Space Block Locking Register (DFLASH_LML)	<a href="#">on page 875</a>
0x0008	Reserved	—
0x000C	DFlash Secondary Low/Mid Address Space Block Locking Register (DFLASH_SLL)	<a href="#">on page 879</a>
0x0010	DFlash Low/Mid Address Space Block Select Register (DFLASH_LMS)	<a href="#">on page 883</a>
0x0014	Reserved	—
0x0018	DFlash Address Register (DFLASH_ADR)	<a href="#">on page 883</a>
0x001C–0x0038	Reserved	—
0x003C	DFlash User Test 0 register (DFLASH_UT0)	<a href="#">on page 884</a>
0x0040	DFlash User Test 1 register (DFLASH_UT1)	<a href="#">on page 887</a>



Table 442. DFlash registers (continued)

Address offset	Register name	Location
0x0044	DFlash User Test 2 register (DFLASH_UT2)	<a href="#">on page 887</a>
0x0048	DFlash User Multiple Input Signature Register 0 (DFLASH_UMISR0)	<a href="#">on page 888</a>
0x004C	DFlash User Multiple Input Signature Register 1 (DFLASH_UMISR1)	<a href="#">on page 889</a>
0x0050	DFlash User Multiple Input Signature Register 2 (DFLASH_UMISR2)	<a href="#">on page 890</a>
0x0054	DFlash User Multiple Input Signature Register 3 (DFLASH_UMISR3)	<a href="#">on page 891</a>
0x0058	DFlash User Multiple Input Signature Register 4 (DFLASH_UMISR4)	<a href="#">on page 892</a>

In the following some nonvolatile registers are described. Please notice that such entities are not Flip-Flops, but locations of TestFlash or Shadow sectors with a special meaning.

During the flash memory initialization phase, the FPEC reads these nonvolatile registers and updates the corresponding volatile registers. When the FPEC detects ECC double errors in these special locations, it behaves in the following way:

- In case of a failing system locations (configurations, device options, redundancy, embedded firmware), the initialization phase is interrupted and a Fatal Error is flagged.
- In case of failing user locations (protections, censorship, platform flash memory controller, ...), the volatile registers are filled with all '1's and the flash memory initialization ends setting low the PEG bit of the corresponding MCR (CFLASH\_MCR or DFLASH\_MCR).

### 30.5.1 CFlash register description

#### 30.5.1.1 CFlash Module Configuration Register (CFLASH\_MCR)

The CFlash Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Offset: 0x0000

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W	w1c															
Reset	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 490. CFlash Module Configuration Register (CFLASH\_MCR)

Table 443. CFLASH\_MCR field descriptions

Field	Description
EDC	<p><i>Ecc Data Correction</i></p> <p>EDC provides information on previous reads. If an ECC Single Error detection and correction occurred, the EDC bit is set to '1'. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to '1' by the user. In the event of an ECC Double Error detection, this bit will not be set. If EDC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EDC) were not corrected through ECC.</p> <p>0: Reads are occurring normally. 1: An ECC Single Error occurred and was corrected during a previous read.</p>
SIZE	<p>array space SIZE</p> <p>The value of SIZE field is dependent upon the size of the flash memory module; see <a href="#">Table 444</a>.</p>
LAS	<p>Low Address Space</p> <p>The value of the LAS field corresponds to the configuration of the Low Address Space; see <a href="#">Table 445</a>.</p>
MAS	<p>Mid Address Space</p> <p>The value of the MAS field corresponds to the configuration of the Mid Address Space; see <a href="#">Table 446</a>.</p>
EER	<p><i>Ecc event ERror</i></p> <p>EER provides information on previous reads. If an ECC Double Error detection occurred, the EER bit is set to '1'. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to '1' by the user. In the event of an ECC Single Error detection and correction, this bit will not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct.</p> <p>0: Reads are occurring normally. 1: An ECC Double Error occurred during a previous read.</p>

**Table 443. CFLASH\_MCR field descriptions (continued)**

Field	Description
RWE	<p>Read-while-Write event Error</p> <p>RWE provides information on previous reads when a Modify operation is on going. If a RWW Error occurs, the RWE bit is set to '1'. Read-While-Write Error means that a read access to the flash memory Matrix has occurred while the FPEC was performing a program or erase operation or an Array Integrity Check.</p> <p>This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to '1' by the user.</p> <p>If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct.</p> <p>0: Reads are occurring normally. 1: A RWW Error occurred during a previous read.</p>
PEAS	<p><i>Program/Erase Access Space</i></p> <p>PEAS is used to indicate which space is valid for program and erase operations: main array space or shadow/test space.</p> <p>PEAS = 0 indicates that the main address space is active for all flash memory module program and erase operations.</p> <p>PEAS = 1 indicates that the test or shadow address space is active for program and erase. The value in PEAS is captured and held with the first interlock write done for Modify operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes).</p> <p>0: Shadow/Test address space is disabled for program/erase and main address space enabled. 1: Shadow/Test address space is enabled for program/erase and main address space disabled.</p>
DONE	<p><i>modify operation DONE</i></p> <p>DONE indicates if the flash memory module is performing a high voltage operation. DONE is set to 1 on termination of the flash memory module reset.</p> <p>DONE is cleared to 0 just after a 0 to 1 transition of EHV, which initiates a high voltage operation, or after resuming a suspended operation.</p> <p>DONE is set to 1 at the end of program and erase high voltage sequences.</p> <p>DONE is set to 1 (within tPABT or tEABT, equal to P/E Abort Latency) after a 1 to 0 transition of EHV, which aborts a high voltage Program/Erase operation.</p> <p>DONE is set to 1 (within tESUS, time equals to Erase Suspend Latency) after a 0 to 1 transition of ESUS, which suspends an erase operation.</p> <p>0: Flash memory is executing a high voltage operation. 1: Flash memory is not executing a high voltage operation.</p>

Table 443. CFLASH\_MCR field descriptions (continued)

Field	Description
PEG	<p><i>Program/Erase Good</i></p> <p>The PEG bit indicates the completion status of the last flash memory Program or Erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the Program and Erase high voltage operations. Aborting a Program/Erase high voltage operation will cause PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the flash memory module is reset, unless a flash memory initialization error has been detected. The value of PEG is valid only when PGM=1 and/or ERS=1 and after DONE transitions from 0 to 1 due to an abort or the completion of a Program/Erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition. The value in PEG is not valid after a 0 to 1 transition of DONE caused by ESUS being set to logic 1. If Program or Erase are attempted on blocks that are locked, the response will be PEG=1, indicating that the operation was successful, and the content of the block were properly protected from the Program or Erase operation. If a Program operation tries to program at '1' bits that are at '0', the program operation is correctly executed on the new bits to be programmed at '0', but PEG is cleared, indicating that the requested operation has failed. In Array Integrity Check or Margin Read PEG is set to 1 when the operation is completed, regardless the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS0-1. Aborting an Array Integrity Check or a Margin Read operation will cause PEG to be cleared to 0, indicating the sequence failed.</p> <p>0: Program, Erase operation failed or Program, Erase, Array Integrity Check or Margin Mode aborted.</p> <p>1: Program or Erase operation successful or Array Integrity Check or Margin Mode completed.</p>
PGM	<p><i>ProGraM</i></p> <p>PGM is used to set up the flash memory module for a Program operation.</p> <p>A 0 to 1 transition of PGM initiates a Program sequence.</p> <p>A 1 to 0 transition of PGM ends the Program sequence.</p> <p>PGM can be set only under User Mode Read (ERS is low and UT0[AIE] is low).</p> <p>PGM can be cleared by the user only when EHV is low and DONE is high.</p> <p>PGM is cleared on reset.</p> <p>0: Flash memory is not executing a Program sequence.</p> <p>1: Flash memory is executing a Program sequence.</p>
PSUS	<p><i>Program SUSpend</i></p> <p>Write this bit has no effect, but the written data can be read back.</p>
ERS	<p><i>ERaSe</i></p> <p>ERS is used to set up the flash memory module for an erase operation.</p> <p>A 0 to 1 transition of ERS initiates an erase sequence.</p> <p>A 1 to 0 transition of ERS ends the erase sequence.</p> <p>ERS can be set only under User Mode Read (PGM is low and UT0[AIE] is low).</p> <p>ERS can be cleared by the user only when ESUS and EHV are low and DONE is high.</p> <p>ERS is cleared on reset.</p> <p>0: Flash memory is not executing an erase sequence.</p> <p>1: Flash memory is executing an erase sequence.</p>

**Table 443. CFLASH\_MCR field descriptions (continued)**

Field	Description
ESUS	<p><i>Erase SUSpend</i></p> <p>ESUS is used to indicate that the flash memory module is in Erase Suspend or in the process of entering a Suspend state. The flash memory module is in Erase Suspend when ESUS = 1 and DONE = 1.</p> <p>ESUS can be set high only when ERS and EHV are high and PGM is low.</p> <p>A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash memory in Erase Suspend. The flash memory module enters Suspend within <math>t_{ESUS}</math> of this transition.</p> <p>ESUS can be cleared only when DONE and EHV are high and PGM is low.</p> <p>A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the module to Erase.</p> <p>The flash memory module cannot exit Erase Suspend and clear DONE while EHV is low. ESUS is cleared on reset.</p> <p>0: Erase sequence is not suspended. 1: Erase sequence is suspended.</p>
EHV	<p><b>Enable High Voltage</b></p> <p>The EHV bit enables the flash memory module for a high voltage program/erase operation. EHV is cleared on reset.</p> <p>EHV must be set after an interlock write to start a program/erase sequence. EHV may be set under one of the following conditions:</p> <p>Erase (ERS = 1, ESUS = 0, UT0[AIE] = 0) Program (ERS = 0, ESUS = 0, PGM = 1, UT0[AIE] = 0)</p> <p>In normal operation, a 1 to 0 transition of EHV with DONE high and ESUS low terminates the current program/erase high voltage operation.</p> <p>When an operation is aborted, there is a 1 to 0 transition of EHV with DONE low and the eventual Suspend bit low. An abort causes the value of PEG to be cleared, indicating a failing program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. Aborting a high voltage operation will leave the flash memory module addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.</p> <p>EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low.</p> <p>0: Flash memory is not enabled to perform an high voltage operation. 1: Flash memory is enabled to perform an high voltage operation.</p>

**Table 444. Array space size**

SIZE	Array space size
000	128 KB
001	256 KB
010	512 KB
011	1024 KB
100	1536 KB
101	Reserved (2048 KB)

Table 444. Array space size (continued)

SIZE	Array space size
110	64 KB
111	Reserved

Table 445. Low address space configuration

LAS	Low address space sectorization
000	Reserved
001	Reserved
010	32 KB + 2 x 16 KB + 2 x 32 KB + 128 KB
011	Reserved
100	Reserved
101	Reserved
110	4 x 16 KB
111	Reserved

Table 446. Mid address space configuration

MAS	Mid address space sectorization
0	2 x 128 KB or 0 KB
1	Reserved

A number of CFLASH\_MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The flash memory module does not allow the user to write bits simultaneously which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in [Table 447](#).

Table 447. CFLASH\_MCR bits set/clear priority levels

Priority level	CFLASH_MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more CFLASH\_MCR bits simultaneously then only the bit with the lowest priority level is written.

If Stall/Abort-While-Write is enabled and an erase operation is started on one sector while fetching code from another then the following sequence is executed:

- CPU is stalled when flash is unavailable
- PEG flag set (stall case) or reset (abort case)
- Interrupt triggered if enabled

If Stall/Abort-While-Write is used then application software should ignore the setting of the RWE flag. The RWE flag should be cleared after each HV operation.

If Stall/Abort-While-Write is not used the application software should handle RWE error. See [Section 30.8.10: Read-while-write functionality](#).

### 30.5.1.2 CFlash Low/Mid Address Space Block Locking Register (CFLASH\_LML)

The CFlash Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the CFLASH\_SLL register, determine if the block is locked from Program or Erase. An “OR” of CFLASH\_LML and CFLASH\_SLL determine the final lock status.

Offset: 0x0004

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK	
W																

Reset Defined by CFLASH\_NVLML at CFlash Test Sector Address 0x403DE8. This location is user OTP (One Time Programmable). The CFLASH\_NVLML register influences only the R/W bits of the CFLASH\_LML register.

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	LLK					
W																

Reset Defined by CFLASH\_NVLML at CFlash Test Sector Address 0x403DE8. This location is user OTP (One Time Programmable). The CFLASH\_NVLML register influences only the R/W bits of the CFLASH\_LML register.

**Figure 491. CFlash Low/Mid Address Space Block Locking Register (CFLASH\_LML)**

Table 448. CFLASH\_LML field descriptions

Field	Description
LME	<p>Low/Mid address space block Enable</p> <p>This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the CFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written.            1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.</p>
TSLK	<p>Test/Shadow address space block Lock</p> <p>This bit is used to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for Program and Erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until CFLASH_MCR[<b>DONE</b>] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high.</p> <p>0: Test/Shadow Address Space Block is unlocked and can be modified (also if CFLASH_SLL[<b>STSLK</b>] = 0).            1: Test/Shadow Address Space Block is locked and cannot be modified.</p>



**Table 448. CFLASH\_LML field descriptions (continued)**

Field	Description
MLK	<p><i>Mid address space block Lock</i></p> <p>This field is used to lock the blocks of Mid Address Space from Program and Erase. MLK is related to sectors B0F7-6, respectively. A value of 1 in a bit of the MLK field signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the MLK field signifies that the corresponding block is available to receive program and erase pulses. The MLK field is not writable after an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK field is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the MLK field. The MLK field may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the MLK field (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the MLK field will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. MLK is not writable unless LME is high.                      0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_SLL[SMLK] = 0).                      1: Mid Address Space Block is locked and cannot be modified.</p>
LLK	<p><i>Low address space block Lock</i></p> <p>This field is used to lock the blocks of Low Address Space from Program and Erase. LLK[5:0] are related to sectors B0F5-0, respectively. LLK[15:6] are not used for this memory cut. A value of 1 in a bit of the LLK field signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the LLK field signifies that the corresponding block is available to receive program and erase pulses. The LLK field is not writable after an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK field is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK field. The LLK field may be written as a register. Reset will cause the field to go back to its TestFlash block value. The default value of the LLK field (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LLK field will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. Bits LLK[15:6] are read-only and locked at '1'. LLK is not writable unless LME is high.                      0: Low Address Space Block is unlocked and can be modified (also if CFLASH_SLL[SLK] = 0).                      1: Low Address Space Block is locked and cannot be modified.</p>

**30.5.1.2.1 CFlash Nonvolatile Low/Mid Address Space Block Locking Register (CFLASH\_NVLML)**

The CFLASH\_LML register has a related CFlash Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for CFLASH\_LML. During the reset phase of the flash memory module, the CFLASH\_NVLML register content is read and loaded into the CFLASH\_LML.

The CFLASH\_NVLML register is a 64-bit register, of which the 32 most significant bits 63:32 are 'don't care' and are used to manage ECC codes.

Offset: 0x403DE8

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	1	1	1	1	1	1	1	1	1	1	TSLK	1	1	MLK	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	1	1	1	1	LLK					
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 492. CFlash Nonvolatile Low/Mid address space block Locking register (CFLASH\_NVLML)

Table 449. CFLASH\_NVLML field descriptions

Field	Description
LME	<p>Low/Mid address space block Enable</p> <p>This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the CFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written.                      1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.</p>
TSLK	<p>Test/Shadow address space block Lock</p> <p>This bit is used to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for Program and Erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high.</p> <p>0: Test/Shadow Address Space Block is unlocked and can be modified (also if CFLASH_SLL[STSLK] = 0).                      1: Test/Shadow Address Space Block is locked and cannot be modified.</p>

**Table 449. CFLASH\_NVLML field descriptions (continued)**

Field	Description
MLK	<p><i>Mid address space block Lock</i></p> <p>These bits are used to lock the blocks of Mid Address Space from Program and Erase. MLK[1:0] are related to sectors B0F7-6, respectively. A value of 1 in a bit of the MLK register signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the MLK register signifies that the corresponding block is available to receive program and erase pulses. The MLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the MLK registers. The MLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the MLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the MLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. MLK is not writable unless LME is high.                      0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_SLL[SMLK] = 0).                      1: Mid Address Space Block is locked and cannot be modified.</p>
LLK	<p><i>Low address space block Lock</i></p> <p>These bits are used to lock the blocks of Low Address Space from Program and Erase. LLK[5:0] are related to sectors B0F5-0, respectively. LLK[15:6] are not used for this memory cut. A value of 1 in a bit of the LLK register signifies that the corresponding block is locked for Program and Erase. A value of 0 in a bit of the LLK register signifies that the corresponding block is available to receive program and erase pulses. The LLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK registers. The LLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the LLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. Bits LLK[15:6] are read-only and locked at '1'. LLK is not writable unless LME is high.                      0: Low Address Space Block is unlocked and can be modified (also if CFLASH_SLL[SLK] = 0).                      1: Low Address Space Block is locked and cannot be modified.</p>

**30.5.1.3 CFlash High Address Space Block Locking Register (CFLASH\_HBL)**

The CFLASH\_HBL register provides a means to protect blocks from being modified.

Offset: 0x0008

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	Defined by CFLASH_NVHBL															
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	HLK							
W																
Reset	Defined by CFLASH_NVHBL															

Figure 493. CFlash Nonvolatile High Address Space Block Locking Register (CFLASH\_HBL)

Table 450. CFLASH\_HBL field descriptions

Field	Description
HBE	<p><i>High address space Block Enable</i></p> <p>This bit is used to enable the Lock registers (HLK5-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the HBE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE the password 0xB2B22222 must be written to the HBL register.</p> <p>0: High Address Locks are disabled: HLK5-0 cannot be written. 1: High Address Locks are enabled: HLK5-0 can be written.</p>
HLK	<p><i>High address space block Lock</i></p> <p>These bits are used to lock the blocks of High Address Space from Program and Erase. HLK11-8 are not used for this memory cut.</p> <p>A value of 1 in a bit of the HLK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the HLK register signifies that the corresponding block is available to receive Program and Erase pulses.</p> <p>The HLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the HLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the HLK registers. The HLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the HLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the HLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 544 KB Flash module bits HLK11-8 are read-only and locked at 1.</p> <p>HLK is not writable unless HBE is high.</p> <p>0: High Address Space Block is unlocked and can be modified. 1: High Address Space Block is locked and cannot be modified.</p>

**30.5.1.3.1 CFlash Nonvolatile High Address Space Block Locking Register (CFLASH\_NVHBL)**

The CFLASH\_HBL register has a related Nonvolatile High Address Space Block Locking register located in TestFlash that contains the default reset value for CFLASH\_HBL. During the reset phase of the Flash module, the CFLASH\_NVHBL register content is read and loaded into the CFLASH\_HBL.

The CFLASH\_NVHBL register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and eventually used to manage ECC codes.

Offset: 0x403DF0

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	HLK							
W																
Reset	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X

**Figure 494. CFlash Nonvolatile High Address Space Block Locking Register (CFLASH\_NVHBL)**

Table 451. CFLASH\_NVHBL field descriptions

Field	Description
HBE	<p><i>High address space Block Enable</i></p> <p>This bit is used to enable the Lock registers (HLK5-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the HBE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE the password 0xB2B22222 must be written to the HBL register.</p> <p>0: High Address Locks are disabled: HLK5-0 cannot be written. 1: High Address Locks are enabled: HLK5-0 can be written.</p>
HLK	<p><i>High address space block Lock</i></p> <p>These bits are used to lock the blocks of High Address Space from Program and Erase. HLK11-8 are not used for this memory cut.</p> <p>A value of 1 in a bit of the HLK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the HLK register signifies that the corresponding block is available to receive Program and Erase pulses.</p> <p>The HLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the HLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the HLK registers. The HLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the HLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the HLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 544 KB Flash module bits HLK11-8 are read-only and locked at 1.</p> <p>HLK is not writable unless HBE is high.</p> <p>0: High Address Space Block is unlocked and can be modified. 1: High Address Space Block is locked and cannot be modified.</p>

#### 30.5.1.4 CFlash Secondary Low/Mid Address Space Block Locking Register (CFLASH\_SLL)

The CFlash Secondary Low/Mid Address Space Block Locking Register provides an alternative means to protect blocks from being modified. These bits, along with bits in the CFLASH\_LML register, determine if the block is locked from Program or Erase. An “OR” of CFLASH\_LML and CFLASH\_SLL determine the final lock status.

Offset: 0x000C

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK	
W																

Reset Defined by CFLASH\_NVSLI at CFlash Test Sector Address 0x403DF8. This location is user OTP (One Time Programmable). The CFLASH\_NVSLI register influences only the R/W bits of the CFLASH\_SLI register.

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	SLK					
W																

Reset Defined by CFLASH\_NVSLI at CFlash Test Sector Address 0x403DF8. This location is user OTP (One Time Programmable). The CFLASH\_NVSLI register influences only the R/W bits of the CFLASH\_SLI register.

Figure 495. CFlash Secondary Low/mid address space block Locking Register (CFLASH\_SLI)

Table 452. CFLASH\_SLL field descriptions

Field	Description
SLE	<p><i>Secondary Low/mid address space block Enable</i></p> <p>This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the CFLASH_SLL register.</p> <p>0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written.</p> <p>1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.</p>
STSLK	<p><i>Secondary Test/Shadow address space block Lock</i></p> <p>This bit is used as an alternate means to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).</p> <p>A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for Program and Erase.</p> <p>A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The STSLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.</p> <p>0: Test/Shadow Address Space Block is unlocked and can be modified (also if CFLASH_LML[TSLK] = 0).</p> <p>1: Test/Shadow Address Space Block is locked and cannot be modified.</p>



**Table 452. CFLASH\_SLL field descriptions (continued)**

Field	Description
SMK	<p><i>Secondary Mid address space block lock</i></p> <p>These bits are used as an alternate means to lock the blocks of Mid Address Space from Program and Erase.</p> <p>SMK[1:0] are related to sectors B0F7-6, respectively.</p> <p>A value of 1 in a bit of the SMK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the SMK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SMK register is not writable once an interlock write is completed until CFLASH_MCR[<i>DONE</i>] is set at the completion of the requested operation. Likewise, the SMK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SMK registers. The SMK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SMK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SMK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>SMK is not writable unless SLE is high.</p> <p>0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_LML[MLK] = 0).</p> <p>1: Mid Address Space Block is locked and cannot be modified.</p>
SLK	<p><i>Secondary Low address space block lock</i></p> <p>These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase.</p> <p>SLK[5:0] are related to sectors B0F5-0, respectively. SLK[15:6] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until CFLASH_MCR[<i>DONE</i>] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>Bits SLK[15:6] are read-only and locked at '1'.</p> <p>SLK is not writable unless SLE is high.</p> <p>0: Low Address Space Block is unlocked and can be modified (also if CFLASH_LML[LLK] = 0).</p> <p>1: Low Address Space Block is locked and cannot be modified.</p>

### 30.5.1.4.1 CFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register (CFLASH\_NVSLI)

The CFLASH\_SLL register has a related Nonvolatile Secondary Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for SLL. During the reset phase of the flash memory module, the CFLASH\_NVSLI register content is read and loaded into the CFLASH\_SLL.

The CFLASH\_NVSLI register is a 64-bit register, of which the 32 most significant bits 63:32 are 'don't care' and are used to manage ECC codes.

Offset: 0x403DF8

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	1	1	1	1	1	1	1	1	1	1	STSLK	1	1	SMK	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	1	1	1	1	SLK					
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 496. CFlash Nonvolatile Secondary Low/mid address space block Locking register (CFLASH\_NVSLI)

**Table 453. CFLASH\_NVSLI field descriptions**

Field	Description
SLE	<p><i>Secondary Low/mid address space block Enable</i>                      This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.                      This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the CFLASH_SLL register.                      0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written.                      1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.</p>
STSLK	<p><i>Secondary Test/Shadow address space block Lock</i>                      This bit is used as an alternate means to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).                      A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for Program and Erase.                      A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.                      The STSLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.                      Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.                      0: Test/Shadow Address Space Block is unlocked and can be modified (also if CFLASH_LML[TSLK] = 0).                      1: Test/Shadow Address Space Block is locked and cannot be modified.</p>

Table 453. CFLASH\_NVSLI field descriptions (continued)

Field	Description
SMK	<p><i>Secondary Mid address space block lock</i></p> <p>These bits are used as an alternate means to lock the blocks of Mid Address Space from Program and Erase.</p> <p>SMK[1:0] are related to sectors B0F7-6, respectively.</p> <p>A value of 1 in a bit of the SMK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the SMK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SMK register is not writable once an interlock write is completed until CFLASH_MCR[<i>DONE</i>] is set at the completion of the requested operation. Likewise, the SMK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SMK registers. The SMK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SMK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SMK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>SMK is not writable unless SLE is high.</p> <p>0: Mid Address Space Block is unlocked and can be modified (also if CFLASH_LML[<i>MLK</i>] = 0).</p> <p>1: Mid Address Space Block is locked and cannot be modified.</p>
SLK	<p><i>Secondary Low address space block lock</i></p> <p>These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase.</p> <p>SLK[5:0] are related to sectors B0F5-0, respectively. SLK[15:6] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until CFLASH_MCR[<i>DONE</i>] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>Bits SLK[15:6] are read-only and locked at '1'.</p> <p>SLK is not writable unless SLE is high.</p> <p>0: Low Address Space Block is unlocked and can be modified (also if CFLASH_LML[<i>LLK</i>] = 0).</p> <p>1: Low Address Space Block is locked and cannot be modified.</p>

**30.5.1.5 CFlash Low/Mid Address Space Block Select Register (CFLASH\_LMS)**

Offset: 0x00010

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	LSL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 497. CFlash Low/Mid address space block Select register (CFLASH\_LMS)**

The CFLASH\_LMS register provides a means to select blocks to be operated on during erase.

Table 454. CFLASH\_LMS field descriptions

Field	Description
MSL	<p><i>Mid address space block SeLect</i></p> <p>A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. MSL[1:0] are related to sectors B0F7-6, respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding MSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect.</p> <p>0: Mid Address Space Block is unselected for erase. 1: Mid Address Space Block is selected for erase.</p>
LSL	<p><i>Low address space block SeLect</i></p> <p>A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. LSL[5:0] are related to sectors B0F5-0, respectively. LSL[15:6] are not used for this memory cut. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding LSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect. Bits LSL[15:6] are read-only and locked at '0'. 0: Low Address Space Block is unselected for erase. 1: Low Address Space Block is selected for erase.</p>

### 30.5.1.6 CFlash High Address Space Block Select Register (CFLASH\_HBS)

The CFLASH\_HBS register provides a means to select blocks to be operated on during erase.

Offset: 0x00014

Access: Read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	HSL							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 498. CFlash High Address Space Block Select Register (CFLASH\_HBS)

Table 455. CFLASH\_HBS field descriptions

Field	Description
HSL	<p><i>High address space block select</i></p> <p>A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected.</p> <p>HSL11-8 are not used for this memory cut.</p> <p>The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the corresponding HSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect.</p> <p>In the 544 KB Flash module bits HSL11-8 are read-only and locked at 0.</p> <p>0: High Address Space Block is unselected for Erase.                      1: High Address Space Block is selected for Erase.</p>

### 30.5.1.7 CFlash Address Register (CFLASH\_ADR)

Offset: 0x00018

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 499. CFlash Address Register (CFLASH\_ADR)**

The CFLASH\_ADR provides the first failing address in the event module failures (ECC or FPEC) occur or the first address at which an ECC single error correction occurs.

**Table 456. CFLASH\_ADR field descriptions**

Field	Description
AD	<p><i>AD</i>dress 22-3 (Read Only)</p> <p>The Address Register provides the first failing address in the event of ECC error (CFLASH_MCR[EER] = 1) or the first failing address in the event of RWW error (CFLASH_MCR[RWE] = 1), or the address of a failure that may have occurred in a FPEC operation (CFLASH_MCR[PEG] = 0). The Address Register also provides the first address at which an ECC single error correction occurs (CFLASH_MCR[EDC] = 1).</p> <p>The ECC double error detection takes the highest priority, followed by the FPEC error and the ECC single error correction. When accessed CFLASH_ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in <a href="#">Table 457</a>.</p> <p>This address is always a Double Word address that selects 64 bits.</p> <p>In case of a simultaneous ECC Double Error Detection on both Double Words of the same page, bit AD3 will output 0. The same is valid for a simultaneous ECC Single Error Correction on both Double Words of the same page.</p>

**Table 457. CFLASH\_ADR content: priority list**

Priority level	Error flag	CFLASH_ADR content
1	CFLASH_MCR[EER] = 1	Address of first ECC Double Error
2	CFLASH_MCR[RWE] = 1	Address of first RWW Error
3	CFLASH_MCR[PEG] = 0	Address of first FPEC Error
4	CFLASH_MCR[EDC] = 1	Address of first ECC Single Error Correction



**30.5.1.8 CFlash User Test 0 register (CFLASH\_UT0)**

The User Test Registers provide the user with the ability to test features on the flash memory module. The User Test 0 Register allows to control the way in which the flash memory content check is done.

Bits MRE, MRV, AIS, EIE and DSI[7:0] of the User Test 0 Register are not accessible whenever CFLASH\_MCR[DONE] or UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x0003C

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	0	0	0	0	0	0	0	DSI							
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

**Figure 500. CFlash User Test 0 register (CFLASH\_UT0)**

**Table 458. CFLASH\_UT0 field descriptions**

Field	Description
UTE	<i>User Test Enable</i> This status bit gives indication when User Test is enabled. All bits in CFLASH_UT0-2 and CFLASH_UMISR0-4 are locked when this bit is 0. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. For UTE the password 0xF9F99999 must be written to the CFLASH_UT0 register.
DSI	<i>Data Syndrome Input</i> These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. Bits DSI[7:0] correspond to the 8 syndrome bits on a double word. These bits are not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect. 0: The syndrome bit is forced at 0. 1: The syndrome bit is forced at 1.
X	<i>Reserved</i> This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Table 458. CFLASH\_UT0 field descriptions (continued)

Field	Description
MRE	<p>Margin Read Enable</p> <p>MRE enables margin reads to be done. This bit, combined with MRV, enables regular user mode reads to be replaced by margin reads inside the Array Integrity Checks sequences. Margin reads are only active during Array Integrity Checks; Normal User reads are not affected by MRE. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: Margin reads are not enabled 1: Margin reads are enabled.</p>
MRV	<p>Margin Read Value</p> <p>If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: Zero's (programmed) margin reads are requested (if MRE = 1). 1: One's (erased) margin reads are requested (if MRE = 1).</p>
EIE	<p>ECC data Input Enable</p> <p>EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: ECC Logic Check is not enabled. 1: ECC Logic Check is enabled.</p>
AIS	<p>Array Integrity Sequence</p> <p>AIS determines the address sequence to be used during array integrity checks or Margin Read. The default sequence (AIS=0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS=1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. The usage of proprietary sequence is forbidden in Margin Read. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: Array Integrity sequence is proprietary sequence. 1: Array Integrity or f sequence is sequential.</p>
AIE	<p>Array Integrity Enable</p> <p>AIE set to '1' starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (CFLASH_UMISR0-4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if CFLASH_MCR[ERS], CFLASH_MCR[PGM] and CFLASH_MCR[EHV] are all low.</p> <p>0: Array Integrity Checks, Margin Read and ECC Logic Checks are not enabled. 1: Array Integrity Checks, Margin Read and ECC Logic Checks are enabled.</p>
AID	<p>Array Integrity Done</p> <p>AID will be cleared upon an Array Integrity Check being enabled (to signify the operation is on-going). Once completed, AID will be set to indicate that the Array Integrity Check is complete. At this time the MISR (CFLASH_UMISR0-4) can be checked.</p> <p>0: Array Integrity Check is on-going. 1: Array Integrity Check is done.</p>

### 30.5.1.9 CFlash User Test 1 register (CFLASH\_UT1)

The CFLASH\_UT1 register allows to enable the checks on the ECC logic related to the 32 LSB of the Double Word.

The User Test 1 Register is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00040

Access: Read/write

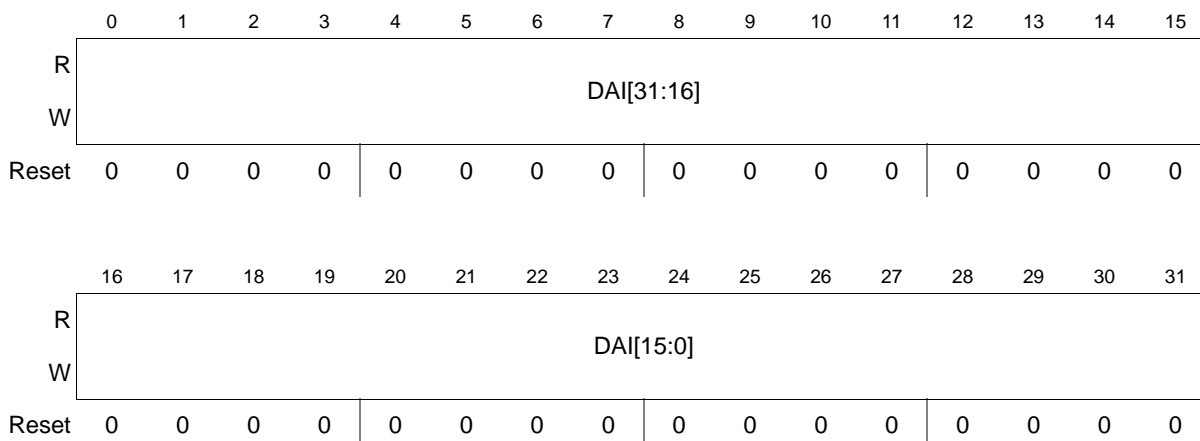


Figure 501. CFlash User Test 1 register (CFLASH\_UT1)

Table 459. CFLASH\_UT1 field descriptions

Field	Description
DAI[31:0]	Data Array Input, bits 31–0 These bits represent the input of even word of ECC logic used in the ECC Logic Check. Bits DAI[31:00] correspond to the 32 array bits representing Word 0 within the double word. 0: The array bit is forced at 0. 1: The array bit is forced at 1.

### 30.5.1.10 CFlash User Test 2 register (CFLASH\_UT2)

The CFLASH\_UT2 register allows to enable the checks on the ECC logic related to the 32 MSB of the Double Word.

The User Test 2 Register is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00044

Access: Read/write

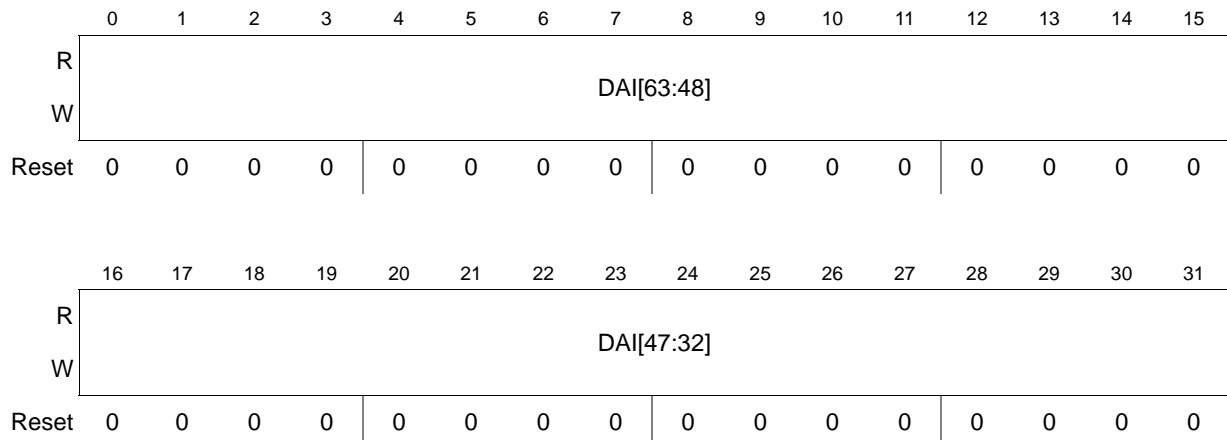


Figure 502. CFlash User Test 2 register (CFLASH\_UT2)

Table 460. CFLASH\_UT2 field descriptions

Field	Description
DAI[63:32]	Data Array Input, bits 63–32 These bits represent the input of odd word of ECC logic used in the ECC Logic Check. Bits DAI[63:32] correspond to the 32 array bits representing Word 1 within the double word. 0: The array bit is forced at 0. 1: The array bit is forced at 1.

**30.5.1.11 CFlash User Multiple Input Signature Register 0 (CFLASH\_UMISR0)**

The CFLASH\_UMISR0 register provides a mean to evaluate the Array Integrity.

The User Multiple Input Signature Register 0 represents the bits 31:0 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH\_UMISR0 Register is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00048

Access: Read/write

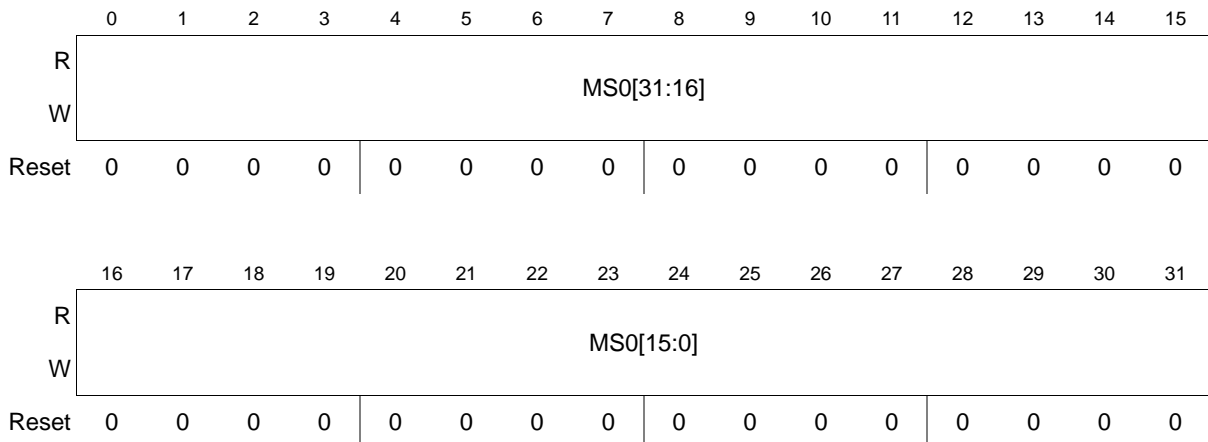


Figure 503. CFlash User Multiple Input Signature Register 0 (CFLASH\_UMISR0)

Table 461. CFLASH\_UMISR0 field descriptions

Field	Description
MS0[31:0]	Multiple input Signature, bits 31–0 These bits represent the MISR value obtained accumulating the bits 31:0 of all the pages read from the flash memory. The MS can be seeded to any value by writing the CFLASH_UMISR0 register.

### 30.5.1.12 CFlash User Multiple Input Signature Register 1 (CFLASH\_UMISR1)

The CFLASH\_UMISR1 provides a means to evaluate the Array Integrity.

The CFLASH\_UMISR1 represents the bits 63:32 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH\_UMISR1 is not accessible whenever CFLASH\_MCR[**DONE**] or CFLASH\_UTO[**AID**] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x0004C

Access: Read/write

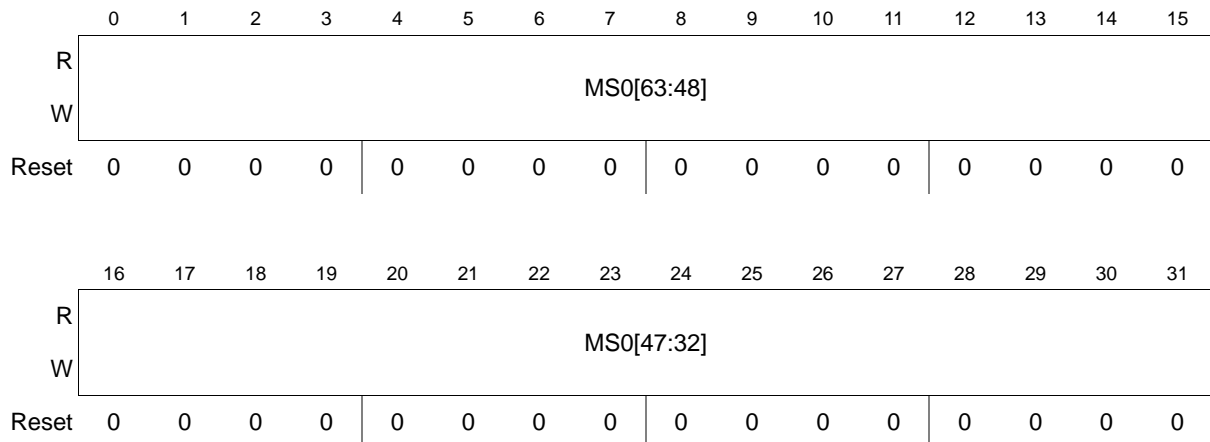


Figure 504. CFlash User Multiple Input Signature Register 1 (CFLASH\_UMISR1)

Table 462. CFLASH\_UMISR1 field descriptions

Field	Description
MS0[63:32]	Multiple input Signature, bits 63–32 These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the flash memory. The MS can be seeded to any value by writing the CFLASH_UMISR1.

### 30.5.1.13 CFlash User Multiple Input Signature Register 2 (CFLASH\_UMISR2)

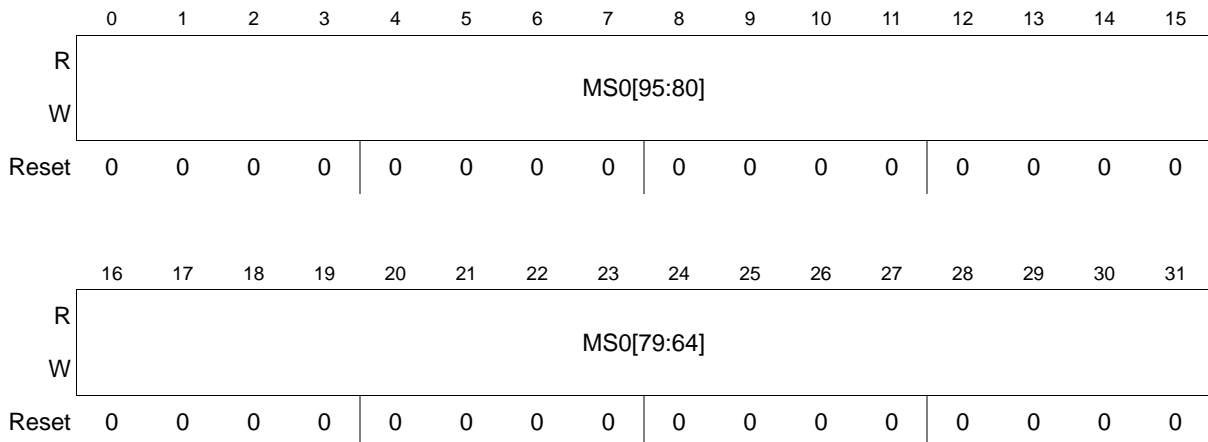
The CFLASH\_UMISR2 provides a means to evaluate the Array Integrity.

The CFLASH\_UMISR2 represents the bits 95:64 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH\_UMISR2 is not accessible whenever CFLASH\_MCR[**DONE**] or CFLASH\_UTO[**AID**] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00050

Access: Read/write



**Figure 505. CFlash User Multiple Input Signature Register 2 (CFLASH\_UMISR2)**

**Table 463. CFLASH\_UMISR2 field descriptions**

Field	Description
MS0[95:64]	Multiple input Signature, bits 95–64 These bits represent the MISR value obtained accumulating the bits 95:64 of all the pages read from the flash memory. The MS can be seeded to any value by writing the CFLASH_UMISR2.

**30.5.1.14 CFlash User Multiple Input Signature Register 3 (CFLASH\_UMISR3)**

The CFLASH\_UMISR3 provides a mean to evaluate the Array Integrity.

The CFLASH\_UMISR3 represents the bits 127:96 of the whole 144 bits word (2 Double Words including ECC).

The CFLASH\_UMISR3 is not accessible whenever CFLASH\_MCR[**DONE**] or CFLASH\_UTO[**AID**] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00054

Access: Read/write

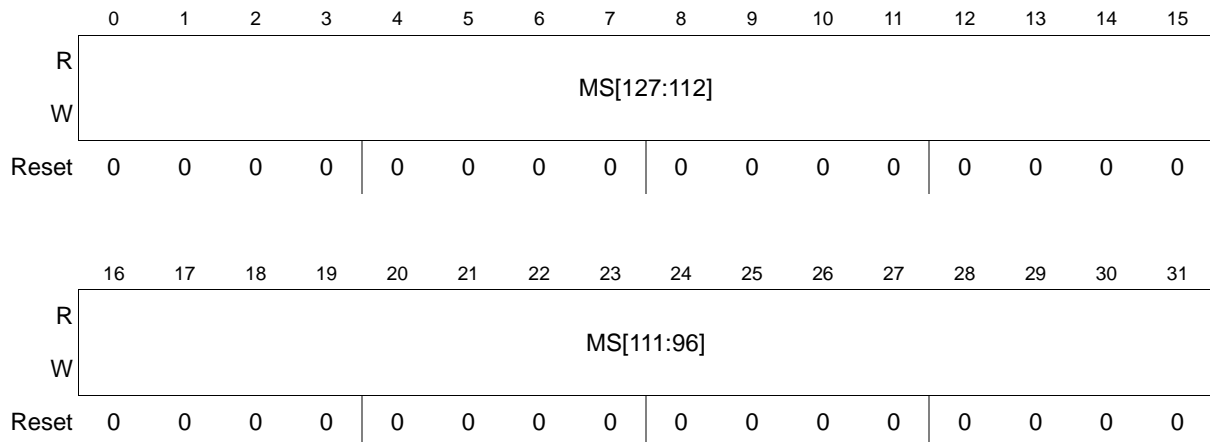


Figure 506. CFlash User Multiple Input Signature Register 3 (CFLASH\_UMISR3)

Table 464. CFLASH\_UMISR3 field descriptions

Field	Description
MS[127:96]	Multiple input Signature, bits127–96 These bits represent the MISR value obtained accumulating the bits 127:96 of all the pages read from the flash memory. The MS can be seeded to any value by writing the CFLASH_UMISR3.

### 30.5.1.15 CFlash User Multiple Input Signature Register 4 (CFLASH\_UMISR4)

The CFLASH\_UMISR4 provides a mean to evaluate the Array Integrity.

The CFLASH\_UMISR4 represents the ECC bits of the whole 144 bits word (2 Double Words including ECC): bits 8:15 are ECC bits for the odd Double Word and bits 24:31 are the ECC bits for the even Double Word; bits 4:5 and 20:21 of MISR are respectively the double and single ECC error detection for odd and even Double Word.

The CFLASH\_UMISR4 is not accessible whenever CFLASH\_MCR[**DONE**] or CFLASH\_UT0[**AID**] are low: reading returns indeterminate data while writing has no effect.



Offset: 0x00058

Access: Read/write

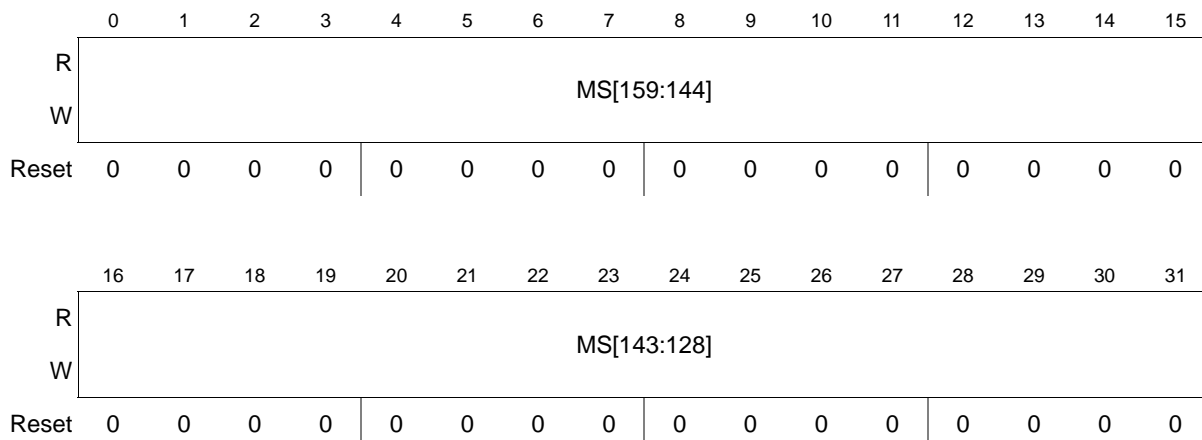


Figure 507. CFlash User Multiple Input Signature Register 4 (CFLASH\_UMISR4)

Table 465. CFLASH\_UMISR4 field descriptions

Field	Description
MS[159:128]	Multiple input Signature, bits 159–128 These bits represent the MISR value obtained accumulating: the 8 ECC bits for the even Double Word (on MS[135:128]); the single ECC error detection for even Double Word (on MS138); the double ECC error detection for even Double Word (on MS139); the 8 ECC bits for the odd Double Word (on MS[151:144]); the single ECC error detection for odd Double Word (on MS154); the double ECC error detection for odd Double Word (on MS155). The MS can be seeded to any value by writing the CFLASH_UMISR4 register.

30.5.1.16 CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0)

The nonvolatile private censorship password 0 register contains the 32 LSB of the Password used to validate the Censorship information contained in NVSCC0–1 registers.

Offset: 0x203DD8

Access: Read/write

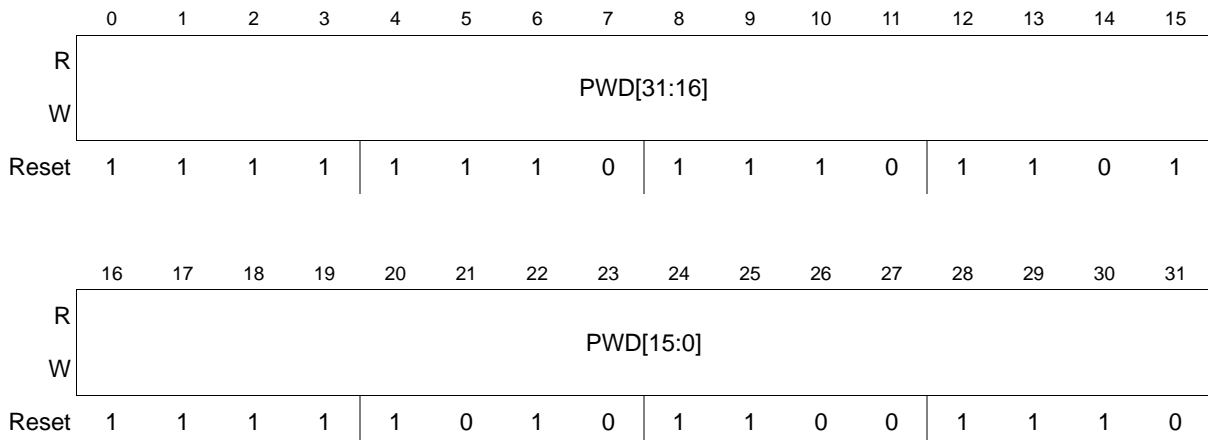


Figure 508. CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0)

Table 466. NVPWD0 field descriptions

Field	Description
PWD[31:0]	Password, bits 31–0 These bits represent the 32 LSB of the Private Censorship Password.

30.5.1.17 CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1)

The nonvolatile private censorship password 1 register contains the 32 MSB of the Password used to validate the Censorship information contained in NVSCC0–1 registers.

Offset: 0x203DDC

Access: Read/write

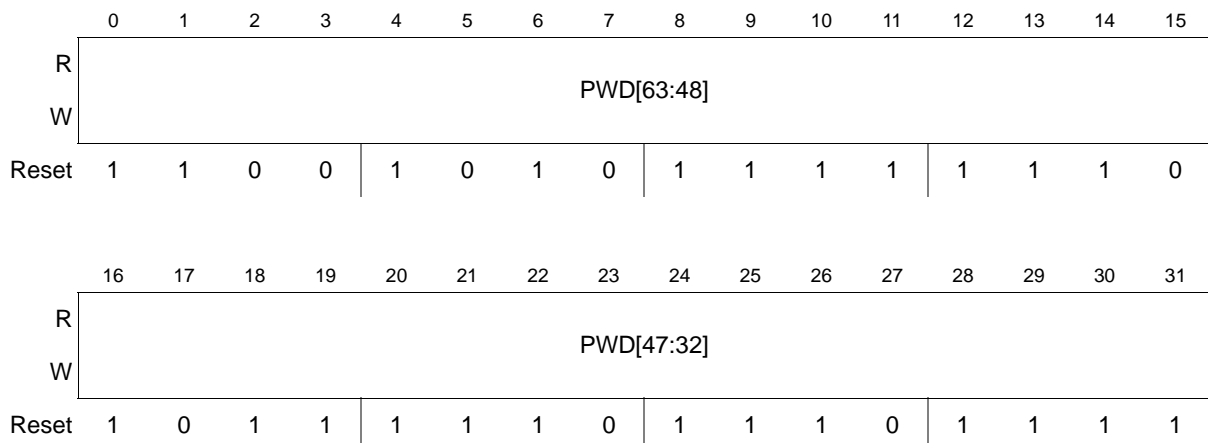


Figure 509. CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1)

**Table 467. NVPWD1 field descriptions**

Field	Description
PWD[63:32]	Password, bits 63–32 These bits represent the 32 MSB of the Private Censorship Password.

*Note:* In a secured device, starting with a serial boot, it is possible to read the content of the four flash locations where the RCHW can be stored. For example if the RCHW is stored at address 0x00000000, the reads at address 0x00000000, 0x00000004, 0x00000008 and 0x0000000C will return a correct value. Any other flash address cannot be accessed.

**30.5.1.18 CFlash Nonvolatile System Censorship Control 0 register (NVSCC0)**

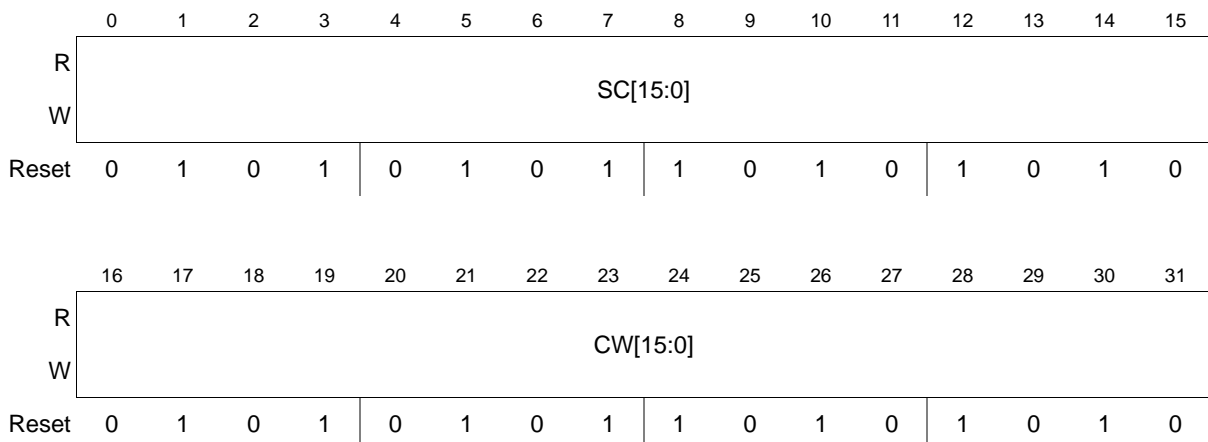
The NVSCC0 register stores the 32 LSB of the Censorship Control Word of the device.

The NVSCC0 is a nonvolatile register located in the Shadow sector: it is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user.

Offset: 0x203DE0

Access: Read/write



**Figure 510. CFlash Nonvolatile System Censorship Control 0 register (NVSCC0)**

**Table 468. NVSCC0 field descriptions**

Field	Description
SC[15:0]	Serial Censorship control word, bits 15-0 These bits represent the 16 LSB of the Serial Censorship Control Word (SCCW). If SC15-0 = 0x55AA and NVSCC1 = NVSCC0 the Public Access is disabled. If SC15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Public Access is enabled.
CW[15:0]	Censorship control Word, bits 15-0 These bits represent the 16 LSB of the Censorship Control Word (CCW). If CW15-0 = 0x55AA and NVSCC1 = NVSCC0 the Censored Mode is disabled. If CW15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Censored Mode is enabled.

### 30.5.1.19 CFlash Nonvolatile System Censorship Control 1 register (NVSCC1)

The NVSCC1 register stores the 32 MSB of the Censorship Control Word of the device.

The NVSCC1 is a nonvolatile register located in the Shadow sector: it is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user.

Offset: 0x203DE4

Access: Read/write

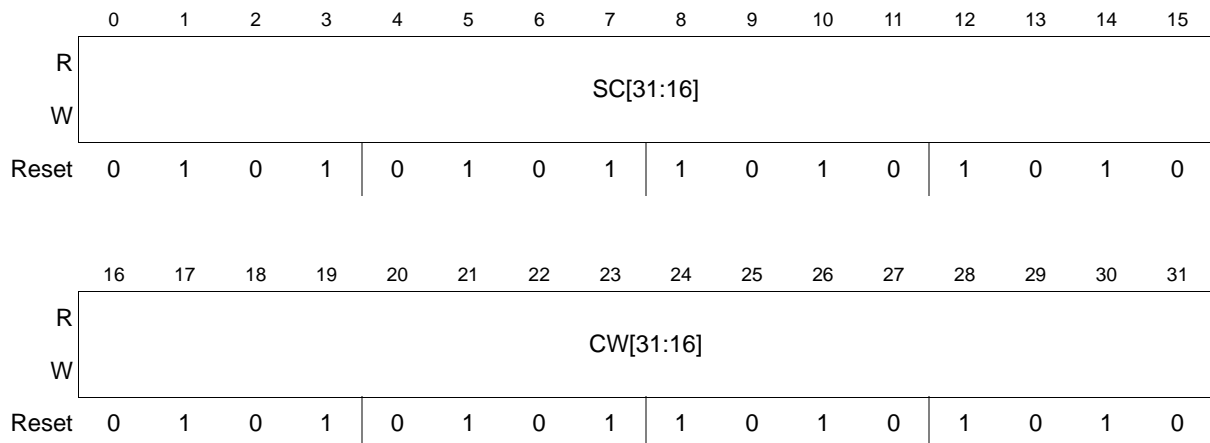


Figure 511. CFlash Nonvolatile System Censorship Control 1 register (NVSCC1)

Table 469. NVSCC1 field descriptions

Field	Description
SC[31:16]	Serial Censorship control word, bits 31-16 These bits represent the 16 MSB of the Serial Censorship Control Word (SCCW). If SC15-0 = 0x55AA and NVSCC1 = NVSCC0 the Public Access is disabled. If SC15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Public Access is enabled.
CW[31:16]	Censorship control Word, bits 31-16 These bits represent the 16 MSB of the Censorship Control Word (CCW). If CW15-0 = 0x55AA and NVSCC1 = NVSCC0 the Censored Mode is disabled. If CW15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Censored Mode is enabled.

### 30.5.1.20 CFlash Nonvolatile User Options register (NVUSRO)

The nonvolatile User Options Register contains configuration information for the user application.

The NVUSRO register is a 64-bit register, of which the 32 most significant bits 63:32 are 'don't care' and are used to manage ECC codes.

Offset: 0x203E18

Access: Read/write

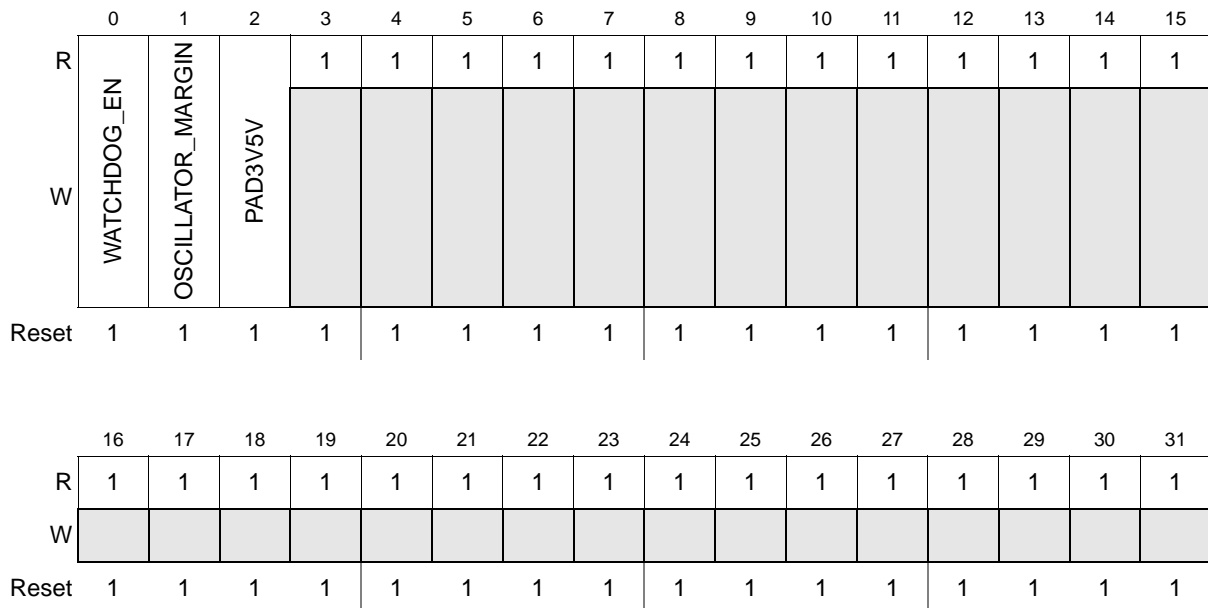


Figure 512. CFlash Nonvolatile User Options register (NVUSRO)

Table 470. NVUSRO field descriptions

Field	Description
WATCHDOG_EN	WATCHDOG_EN 0: Disable after reset 1: Enable after reset Default manufacturing value before flash memory initialization is '1'
OSCILLATOR_MARGIN	OSCILLATOR_MARGIN 0: Low consumption configuration (4 MHz/8 MHz) 1: High margin configuration (4 MHz/16 MHz) Default manufacturing value before flash memory initialization is '1'
PAD3V5V	PAD3V5V 0: High voltage supply is 5.0 V 1: High voltage supply is 3.3 V Default manufacturing value before flash memory initialization is '1' (3.3 V) which should ensure correct minimum slope for boundary scan.

### 30.5.2 DFlash register description

#### 30.5.2.1 DFlash Module Configuration Register (DFLASH\_MCR)

The Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Address offset: 0x0000

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W	w1c															
Reset	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 513. DFlash Module Configuration Register (DFLASH\_MCR)

Table 471. DFLASH\_MCR field descriptions

Field	Description
EDC	ECC Data Correction EDC provides information on previous reads. If an ECC Single Error detection and correction occurred, the EDC bit is set to '1'. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to '1' by the user. In the event of an ECC Double Error detection, this bit will not be set. If EDC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EDC) were not corrected through ECC. The function of this bit is device dependent and it can be configured to be disabled. 0: Reads are occurring normally. 1: An ECC Single Error occurred and was corrected during a previous read.
SIZE	array space SIZE The value of SIZE field is dependent upon the size of the flash memory module; see <a href="#">Table 472</a> .
LAS	Low Address Space The value of the LAS field corresponds to the configuration of the Low Address Space; see <a href="#">Table 473</a> .
MAS	Mid Address Space The value of the MAS field corresponds to the configuration of the Mid Address Space; see <a href="#">Table 474</a> .
EER	ECC event Error EER provides information on previous reads. If an ECC Double Error detection occurred, the EER bit is set to '1'. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to '1' by the user. In the event of an ECC Single Error detection and correction, this bit will not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct. 0: Reads are occurring normally. 1: An ECC Double Error occurred during a previous read.

**Table 471. DFLASH\_MCR field descriptions (continued)**

Field	Description
RWE	<p>Read-while-Write event Error</p> <p>RWE provides information on previous reads when a Modify operation is on going. If a RWW Error occurs, the RWE bit will be set to '1'. Read-While-Write Error means that a read access to the flash memory Matrix has occurred while the FPEC was performing a program or erase operation or an Array Integrity Check.</p> <p>This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to '1' by the user.</p> <p>If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct.</p> <p>0: Reads are occurring normally. 1: A RWW Error occurred during a previous read.</p>
PEAS	<p>Program/Erase Access Space</p> <p>PEAS is used to indicate which space is valid for program and erase operations: main array space or shadow/test space.</p> <p>PEAS = 0 indicates that the main address space is active for all flash memory module program and erase operations.</p> <p>PEAS = 1 indicates that the test or shadow address space is active for program and erase. The value in PEAS is captured and held with the first interlock write done for Modify operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes).</p> <p>0: Shadow/Test address space is disabled for program/erase and main address space enabled. 1: Shadow/Test address space is enabled for program/erase and main address space disabled.</p>
DONE	<p>modify operation DONE</p> <p>DONE indicates if the flash memory module is performing a high voltage operation. DONE is set to 1 on termination of the flash memory module reset.</p> <p>DONE is cleared to 0 just after a 0 to 1 transition of EHV, which initiates a high voltage operation, or after resuming a suspended operation.</p> <p>DONE is set to 1 at the end of program and erase high voltage sequences.</p> <p>DONE is set to 1 (within tPABT or tEABT, equal to P/E Abort Latency) after a 1 to 0 transition of EHV, which aborts a high voltage Program/Erase operation.</p> <p>DONE is set to 1 (within tESUS, time equals to Erase Suspend Latency) after a 0 to 1 transition of ESUS, which suspends an erase operation.</p> <p>0: Flash memory is executing a high voltage operation. 1: Flash memory is not executing a high voltage operation.</p>

Table 471. DFLASH\_MCR field descriptions (continued)

Field	Description
PEG	<p>Program/Erase Good</p> <p>The PEG bit indicates the completion status of the last flash memory program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations.</p> <p>Aborting a program/erase high voltage operation will cause PEG to be cleared to '0', indicating the sequence failed.</p> <p>PEG is set to '1' when the flash memory module is reset, unless a flash memory initialization error has been detected.</p> <p>The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition.</p> <p>The value in PEG is not valid after a 0 to 1 transition of DONE caused by ESUS being set to logic 1.</p> <p>If program or erase are attempted on blocks that are locked, the response will be PEG = 1, indicating that the operation was successful, and the content of the block were properly protected from the program or erase operation.</p> <p>If a Program operation tries to program at '1' bits that are at '0', the program operation is correctly executed on the new bits to be programmed at '0', but PEG is cleared, indicating that the requested operation has failed.</p> <p>In Array Integrity Check or Margin Read PEG is set to 1 when the operation is completed, regardless the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS0-1.</p> <p>Aborting an Array Integrity Check or a Margin Read operation will cause PEG to be cleared to 0, indicating the sequence failed.</p> <p>0: Program, Erase operation failed or Program, Erase, Array Integrity Check or Maring Mode aborted.</p> <p>1: Program or Erase operation succesful or Array Integrity Check or Maring Mode completed.</p>
PGM	<p>ProGraM</p> <p>PGM is used to set up the flash memory module for a Program operation.</p> <p>A 0 to 1 transition of PGM initiates a Program sequence.</p> <p>A 1 to 0 transition of PGM ends the Program sequence.</p> <p>PGM can be set only under User Mode Read (ERS is low and DFLASH_UT0[AIE] is low).</p> <p>PGM can be cleared by the user only when EHV is low and DONE is high.</p> <p>PGM is cleared on reset.</p> <p>0: Flash memory is not executing a Program sequence.</p> <p>1: Flash memory is executing a Program sequence.</p>
PSUS	<p><b>PSUS:</b> <i>Program SUSpend</i></p> <p>Write this bit has no effect, but the written data can be read back.</p>
ERS	<p>ERaSe</p> <p>ERS is used to set up the flash memory module for an erase operation.</p> <p>A 0 to 1 transition of ERS initiates an erase sequence.</p> <p>A 1 to 0 transition of ERS ends the erase sequence.</p> <p>ERS can be set only under User Mode Read (PGM is low and DFLASH_UT0[AIE] is low).</p> <p>ERS can be cleared by the user only when ESUS and EHV are low and DONE is high.</p> <p>ERS is cleared on reset.</p> <p>0: Flash memory is not executing an erase sequence.</p> <p>1: Flash memory is executing an erase sequence.</p>



**Table 471. DFLASH\_MCR field descriptions (continued)**

Field	Description
ESUS	<p>Erase SUSpend</p> <p>ESUS is used to indicate that the flash memory module is in Erase Suspend or in the process of entering a Suspend state. The flash memory module is in Erase Suspend when ESUS = 1 and DONE = 1.</p> <p>ESUS can be set high only when ERS and EHV are high and PGM is low.</p> <p>A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash memory in Erase Suspend. The flash memory module enters Suspend within <math>t_{ESUS}</math> of this transition.</p> <p>ESUS can be cleared only when DONE and EHV are high and PGM is low.</p> <p>A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the module to Erase.</p> <p>The flash memory module cannot exit Erase Suspend and clear DONE while EHV is low.</p> <p>ESUS is cleared on reset.</p> <p>0: Erase sequence is not suspended. 1: Erase sequence is suspended.</p>
EHV	<p>Enable High Voltage</p> <p>The EHV bit enables the flash memory module for a high voltage program/erase operation. EHV is cleared on reset.</p> <p>EHV must be set after an interlock write to start a program/erase sequence. EHV may be set under one of the following conditions:</p> <p>Erase (ERS = 1, ESUS = 0, DFLASH_UT0[AIE] = 0)</p> <p>Program (ERS = 0, ESUS = 0, PGM = 1, DFLASH_UT0[AIE] = 0)</p> <p>In normal operation, a 1 to 0 transition of EHV with DONE high and ESUS low terminates the current program/erase high voltage operation.</p> <p>When an operation is aborted, there is a 1 to 0 transition of EHV with DONE low and the eventual Suspend bit low. An abort causes the value of PEG to be cleared, indicating a failing program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. Aborting a high voltage operation will leave the flash memory module addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.</p> <p>EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low.</p> <p>0: Flash memory is not enabled to perform an high voltage operation. 1: Flash memory is enabled to perform an high voltage operation.</p>

**Table 472. Array space size**

SIZE	Array space size
000	128 KB
001	256 KB
010	512 KB
011	Reserved (1024 KB)
100	Reserved (1536 KB)
101	Reserved (2048 KB)

**Table 472. Array space size (continued)**

SIZE	Array space size
110	64 KB
111	Reserved

**Table 473. Low address space configuration**

LAS	Low address space sectorization
000	Reserved
001	Reserved
010	32 KB + 2 x 16 KB + 2 x 32 KB + 128 KB
011	Reserved
100	Reserved
101	Reserved
110	4 x 16 KB
111	Reserved

**Table 474. Mid address space configuration**

MAS	Mid address space sectorization
0	2 x 128KB
1	Reserved

A number of DFLASH\_MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The flash memory module does not allow the user to write bits simultaneously which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in the [Table 475](#).

**Table 475. DFLASH\_MCR bits set/clear priority levels**

Priority level	DFLASH_MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more DFLASH\_MCR bits simultaneously then only the bit with the lowest priority level is written.

If Stall/Abort-While-Write is enabled and an erase operation is started on one sector while fetching code from another then the following sequence is executed:

- CPU is stalled when flash is unavailable
- PEG flag set (stall case) or reset (abort case)
- Interrupt triggered if enabled

If Stall/Abort-While-Write is used then application software should ignore the setting of the RWE flag. The RWE flag should be cleared after each HV operation.

If Stall/Abort-While-Write is not used the application software should handle RWE error. See [Section 30.8.10: Read-while-write functionality](#).

### 30.5.2.2 DFlash Low/Mid Address Space Block Locking Register (DFLASH\_LML)

The DFlash Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the DFLASH\_SLL register, determine if the block is locked from Program or Erase. An “OR” of DFLASH\_LML and DFLASH\_SLL determine the final lock status.

Offset: 0x0004

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	0	0
W																

Reset Defined by DFLASH\_NVLML at DFlash Test Sector Address 0xC03DE8. This location is user OTP (One Time Programmable). The DFLASH\_NVLML register influences only the R/W bits of the DFLASH\_LML register.

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	LLK			
W																

Reset Defined by DFLASH\_NVLML at DFlash Test Sector Address 0xC03DE8. This location is user OTP (One Time Programmable). The DFLASH\_NVLML register influences only the R/W bits of the DFLASH\_LML register.

**Figure 514. DFlash Low/Mid Address Space Block Locking Register (DFLASH\_LML)**

Table 476. DFLASH\_LML field descriptions

Field	Description
LME	<p>Low/Mid address space block Enable</p> <p>This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the DFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written. 1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.</p>
TSLK	<p>Test/Shadow address space block Lock</p> <p>This bit is used to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for Program and Erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high.</p> <p>0: Test/Shadow Address Space Block is unlocked and can be modified (also if DFLASH_SLL[STSLK] = 0). 1: Test/Shadow Address Space Block is locked and cannot be modified.</p>
LLK	<p>Low address space block Lock</p> <p>This field is used to lock the blocks of Low Address Space from Program and Erase. LLK[3:0] are related to sectors B1F3-0, respectively. LLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the LLK field signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the LLK field signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The LLK field is not writable after an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK field is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the LLK field. The LLK field may be written as a register. Reset will cause the field to go back to its TestFlash block value. The default value of the LLK field (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the LLK field will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 64 KB flash memory module bits LLK[15:4] are read-only and locked at '1'. LLK is not writable unless LME is high.</p> <p>0: Low Address Space Block is unlocked and can be modified (also if DFLASH_SLL[SLK] = 0). 1: Low Address Space Block is locked and cannot be modified.</p>

**30.5.2.2.1 DFlash Nonvolatile Low/Mid Address Space Block Locking Register (DFLASH\_NVLML)**

The DFLASH\_LML register has a related Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for DFLASH\_LML. During the reset phase of the flash memory module, the DFLASH\_NVLML register content is read and loaded into the DFLASH\_LML.

The DFLASH\_NVLML register is a 64-bit register, of which the 32 most significant bits 63:32 are ‘don’t care’ and are used to manage ECC codes.

Offset: 0xC03DE8

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	1	1	1	1	1	1	1	1	1	1	TSLK	1	1	1	1
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	1	1	1	1	1	1	LLK			
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 515. DFlash Nonvolatile Low/Mid address space block Locking register (DFLASH\_NVLML)**

Table 477. DFLASH\_NVLM field descriptions

Field	Description
LME	<p>Low/Mid address space block Enable</p> <p>This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the DFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written.  1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.</p>
TSLK	<p>Test/Shadow address space block Lock</p> <p>This bit is used to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for Program and Erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high.</p> <p>0: Test/Shadow Address Space Block is unlocked and can be modified (also if DFLASH_SLL[STSLK] = 0).  1: Test/Shadow Address Space Block is locked and cannot be modified.</p>
LLK	<p>Low address space block Lock</p> <p>These bits are used to lock the blocks of Low Address Space from Program and Erase. LLK[3:0] are related to sectors B1F3-0, respectively. LLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the LLK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the LLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The LLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the LLK registers. The LLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the LLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the LLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 64 KB flash memory module bits LLK[15:4] are read-only and locked at '1'. LLK is not writable unless LME is high.</p> <p>0: Low Address Space Block is unlocked and can be modified (also if DFLASH_SLL[SLK] = 0).  1: Low Address Space Block is locked and cannot be modified.</p>

**30.5.2.3 DFlash Secondary Low/Mid Address Space Block Locking Register (DFLASH\_SLL)**

The DFlash Secondary Low/Mid Address Space Block Locking Register provides an alternative means to protect blocks from being modified. These bits, along with bits in the DFLASH\_LML register, determine if the block is locked from Program or Erase. An “OR” of DFLASH\_LML and DFLASH\_SLL determine the final lock status.

Offset: 0x000C

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	0	0
W																

Reset Defined by DFLASH\_NVSLI at DFlash Test Sector Address 0xC03DF8. This location is user OTP (One Time Programmable). The DFLASH\_NVSLI register influences only the R/W bits of the DFLASH\_SLL register.

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	SLK			
W																

Reset Defined by DFLASH\_NVSLI at DFlash Test Sector Address 0xC03DF8. This location is user OTP (One Time Programmable). The DFLASH\_NVSLI register influences only the R/W bits of the DFLASH\_SLL register.

**Figure 516. DFlash Secondary Low/mid address space block Locking register (DFLASH\_SLL)**

Table 478. DFLASH\_SLL field descriptions

Field	Description
SLE	<p><i>Secondary Low/mid address space block Enable</i></p> <p>This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the DFLASH_SLL register.</p> <p>0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written.</p> <p>1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.</p>
STSLK	<p><i>Secondary Test/Shadow address space block Lock</i></p> <p>This bit is used as an alternate means to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).</p> <p>A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for Program and Erase.</p> <p>A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The STSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.</p> <p>0: Test/Shadow Address Space Block is unlocked and can be modified (also if DFLASH_LML[TSLK] = 0).</p> <p>1: Test/Shadow Address Space Block is locked and cannot be modified.</p>
SLK	<p><i>Secondary Low address space block lock</i></p> <p>These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase.</p> <p>SLK[3:0] are related to sectors B1F3-0, respectively. SLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 64 KB flash memory module bits SLK[15:4] are read-only and locked at '1'.</p> <p>SLK is not writable unless SLE is high.</p> <p>0: Low Address Space Block is unlocked and can be modified (also if DFLASH_LML[LLK] = 0).</p> <p>1: Low Address Space Block is locked and cannot be modified.</p>



**30.5.2.3.1 DFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register (DFLASH\_NVSL)**

The DFLASH\_SLL register has a related Nonvolatile Secondary Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for DFLASH\_SLL. During the reset phase of the flash memory module, the DFLASH\_NVSL register content is read and loaded into the DFLASH\_SLL.

The DFLASH\_NVSL register is a 64-bit register, of which the 32 most significant bits 63:32 are 'don't care' and are used to manage ECC codes.

Offset: 0xC03DF8

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	1	1	1	1	1	1	1	1	1	1	STSLK	1	1	1	1
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	1	1	1	1	1	1	SLK			
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 517. DFlash Nonvolatile Secondary Low/mid address space block Locking register (DFLASH\_NVSL)**

Table 479. DFLASH\_NVSLI field descriptions

Field	Description
SLE	<p><i>Secondary Low/mid address space block Enable</i></p> <p>This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the DFLASH_SLL register.</p> <p>0: Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written.</p> <p>1: Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.</p>
STSLK	<p><i>Secondary Test/Shadow address space block Lock</i></p> <p>This bit is used as an alternate means to lock the block of Test and Shadow Address Space from Program and Erase (Erase is any case forbidden for Test block).</p> <p>A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for Program and Erase.</p> <p>A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The STSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.</p> <p>0: Test/Shadow Address Space Block is unlocked and can be modified (also if DFLASH_LML[TSLK] = 0).</p> <p>1: Test/Shadow Address Space Block is locked and cannot be modified.</p>
SLK	<p><i>Secondary Low address space block lock</i></p> <p>These bits are used as an alternate means to lock the blocks of Low Address Space from Program and Erase.</p> <p>SLK[3:0] are related to sectors B1F3-0, respectively. SLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for Program and Erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. In the 64 KB flash memory module bits SLK[15:4] are read-only and locked at '1'.</p> <p>SLK is not writable unless SLE is high.</p> <p>0: Low Address Space Block is unlocked and can be modified (also if DFLASH_LML[LLK] = 0).</p> <p>1: Low Address Space Block is locked and cannot be modified.</p>

### 30.5.2.4 DFlash Low/Mid Address Space Block Select Register (DFLASH\_LMS)

The DFLASH\_LMS register provides a means to select blocks to be operated on during erase.

Offset: 0x00010

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	LSL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 518. DFlash Low/Mid Address Space Block Select Register (DFLASH\_LMS)

Table 480. DFLASH\_LMS field descriptions

Field	Description
LSL	<p>Low address space block SeLect</p> <p>A value of 1 in the select register signifies that the block is selected for erase.</p> <p>A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected.</p> <p>LSL[3:0] are related to sectors B1F3-0, respectively. LSL[15:4] are not used for this memory cut.</p> <p>The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the corresponding LSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect.</p> <p>In the 80 KB flash memory module bits LSL[15:4] are read-only and locked at '0'.</p> <p>0: Low Address Space Block is unselected for Erase.</p> <p>1: Low Address Space Block is selected for Erase.</p>

### 30.5.2.5 DFlash Address Register (DFLASH\_ADR)

The DFLASH\_ADR provides the first failing address in the event module failures (ECC, RWW or FPEC) occur or the first address at which an ECC single error correction occurs.

Address offset: 0x00018

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 519. DFlash Address Register (DFLASH\_ADR)

Table 481. DFLASH\_ADR field descriptions

Field	Description
AD[22:3]	<p>Address 22-3</p> <p>The Address Register provides the first failing address in the event of ECC error (DFLASH_MCR[EER] set) or the first failing address in the event of RWW error (DFLASH_MCR[RWE] set), or the address of a failure that may have occurred in a FPEC operation (DFLASH_MCR[PEG] cleared). The Address Register also provides the first address at which an ECC single error correction occurs (DFLASH_MCR[EDC] set), if the device is configured to show this feature.</p> <p>The ECC double error detection takes the highest priority, followed by the RWW error, the FPEC error and the ECC single error correction. When accessed DFLASH_ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in the <a href="#">Table 482</a>.</p> <p>This address is always a Double Word address that selects 64 bits.</p> <p>In case of a simultaneous ECC Double Error Detection on both Double Words of the same page, bit AD3 will output 0. The same is valid for a simultaneous ECC Single Error Correction on both Double Words of the same page.</p> <p>In User Mode the Address Register is read only.</p>

Table 482. DFLASH\_ADR content: priority list

Priority level	Error flag	DFLASH_ADR content
1	DFLASH_MCR[EER] = 1	Address of first ECC Double Error
2	DFLASH_MCR[RWE] = 1	Address of first RWW Error
3	DFLASH_MCR[PEG] = 0	Address of first FPEC Error
4	DFLASH_MCR[EDC] = 1	Address of first ECC Single Error Correction

30.5.2.6 DFlash User Test 0 register (DFLASH\_UT0)

The User Test Registers provide the user with the ability to test features on the flash memory module.



The User Test 0 Register allows to control the way in which the flash memory content check is done.

Bits MRE, MRV, AIS, EIE and DSI[7:0] of the User Test 0 Register are not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x0003C

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	0	0	0	0	0	0	0	DSI							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 520. DFlash User Test 0 register (DFLASH\_UT0)

Table 483. DFLASH\_UT0 field descriptions

Field	Description
UTE	<p>User Test Enable</p> <p>This status bit gives indication when User Test is enabled. All bits in DFLASH_UT0-2 and DFLASH_UMISR0-4 are locked when this bit is 0.</p> <p>This bit is not writeable to a 1, but may be cleared. The reset value is 0.</p> <p>The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write.</p> <p>For UTE the password 0xF9F99999 must be written to the DFLASH_UT0 register.</p>
DSI	<p>Data Syndrome Input</p> <p>These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. Bits DSI[7:0] correspond to the 8 syndrome bits on a double word.</p> <p>These bits are not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: The syndrome bit is forced at 0.</p> <p>1: The syndrome bit is forced at 1.</p>
X	<p>Reserved</p> <p>This bit can be written and its value can be read back, but there is no function associated.</p> <p>This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p>

Table 483. DFLASH\_UT0 field descriptions (continued)

Field	Description
MRE	<p>Margin Read Enable</p> <p>MRE enables margin reads to be done. This bit, combined with MRV, enables regular user mode reads to be replaced by margin reads.</p> <p>Margin reads are only active during Array Integrity Checks; Normal User reads are not affected by MRE.</p> <p>This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: Margin reads are not enabled, all reads are User mode reads.</p> <p>1: Margin reads are enabled.</p>
MRV	<p>Margin Read Value</p> <p>If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0).</p> <p>This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: Zero's (programmed) margin reads are requested (if MRE = 1).</p> <p>1: One's (erased) margin reads are requested (if MRE = 1).</p>
EIE	<p>ECC data Input Enable</p> <p>EIE enables the ECC Logic Check operation to be done.</p> <p>This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: ECC Logic Check is not enabled.</p> <p>1: ECC Logic Check is enabled.</p>
AIS	<p>Array Integrity Sequence</p> <p>AIS determines the address sequence to be used during array integrity checks or Margin Read. The default sequence (AIS = 0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary.</p> <p>The alternative sequence (AIS = 1) is just logically sequential. Proprietary sequence is forbidden in Margin Read.</p> <p>It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence.</p> <p>This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0: Array Integrity sequence is proprietary sequence.</p> <p>1: Array Integrity or Margin Read sequence is sequential.</p>
AIE	<p>Array Integrity Enable</p> <p>AIE set to '1' starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (DFLASH_UMISR0-4) can be checked after the operation is complete, to determine if a correct signature is obtained.</p> <p>AIE can be set only if DFLASH_MCR[ERS], DFLASH_MCR[PGM] and DFLASH_MCR[EHV] are all low.</p> <p>0: Array Integrity Checks are not enabled.</p> <p>1: Array Integrity Checks are enabled.</p>
AID	<p>Array Integrity Done</p> <p>AID will be cleared upon an Array Integrity Check being enabled (to signify the operation is on-going). Once completed, AID will be set to indicate that the Array Integrity Check is complete. At this time the MISR (DFLASH_UMISR0-4) can be checked.</p> <p>0: Array Integrity Check is on-going.</p> <p>1: Array Integrity Check is done.</p>

### 30.5.2.7 DFlash User Test 1 register (DFLASH\_UT1)

The DFLASH\_UT1 register allows to enable the checks on the ECC logic related to the 32 LSB of the Double Word.

The User Test 1 Register is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x00040

Access: Read/write

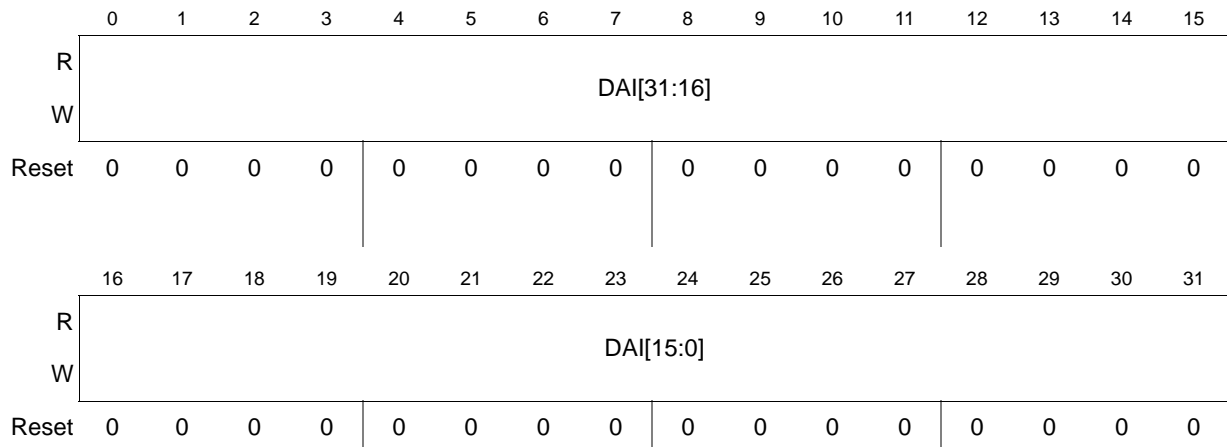


Figure 521. DFlash User Test 1 register (DFLASH\_UT1)

Table 484. DFLASH\_UT1 field descriptions

Field	Description
DAI[31:16]	Data Array Input, bits 31-0 These bits represent the input of even word of ECC logic used in the ECC Logic Check. Bits DAI[31:00] correspond to the 32 array bits representing Word 0 within the double word. 0: The array bit is forced at 0. 1: The array bit is forced at 1.

### 30.5.2.8 DFlash User Test 2 register (DFLASH\_UT2)

The DFLASH\_UT2 register allows to enable the checks on the ECC logic related to the 32 MSB of the Double Word.

The User Test 2 Register is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00044

Reset value: 0x0000\_0000

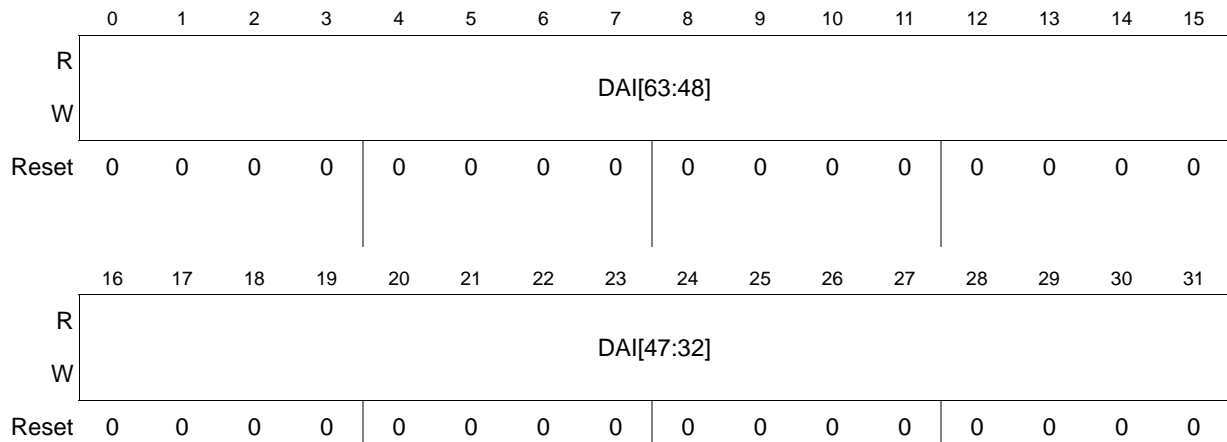


Figure 522. DFlash User Test 2 register (DFLASH\_UT2)

Table 485. DFLASH\_UT2 field descriptions

Field	Description
DAI[63:32]	Data Array Input, bits 63-32 These bits represent the input of odd word of ECC logic used in the ECC Logic Check. Bits DAI[63:32] correspond to the 32 array bits representing Word 1 within the double word. 0: The array bit is forced at 0. 1: The array bit is forced at 1.

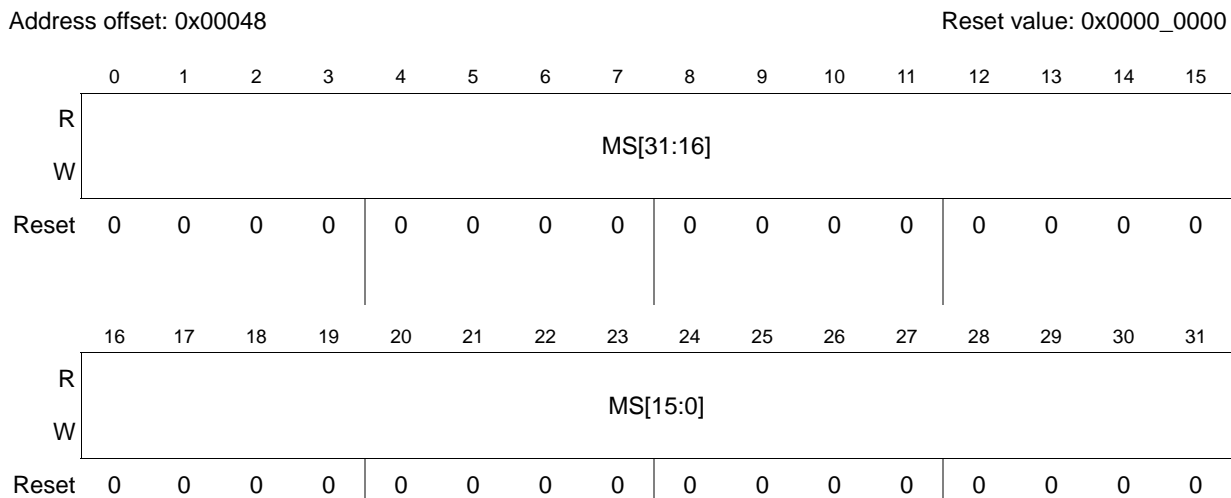
### 30.5.2.9 DFlash User Multiple Input Signature Register 0 (DFLASH\_UMISR0)

The DFLASH\_UMISR0 provides a means to evaluate the Array Integrity.

The DFLASH\_UMISR0 represents the bits 31:0 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH\_UMISR0 is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.





**Figure 523. DFlash User Multiple Input Signature Register 0 (DFLASH\_UMISR0)**

**Table 486. DFLASH\_UMISR0 field descriptions**

Field	Description
MS[31:0]	Multiple input Signature, bits 31–0 These bits represent the MISR value obtained accumulating the bits 31:0 of all the pages read from the flash memory. The MS can be seeded to any value by writing the DFLASH_UMISR0 register.

**30.5.2.10 DFlash User Multiple Input Signature Register 1 (DFLASH\_UMISR1)**

The DFLASH\_UMISR1 provides a mean to evaluate the Array Integrity.

The DFLASH\_UMISR1 represents the bits 63:32 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH\_UMISR1 is not accessible whenever DFLASH\_MCR[**DONE**] or DFLASH\_UT0[**AID**] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x0004C

Reset value: 0x0000\_0000

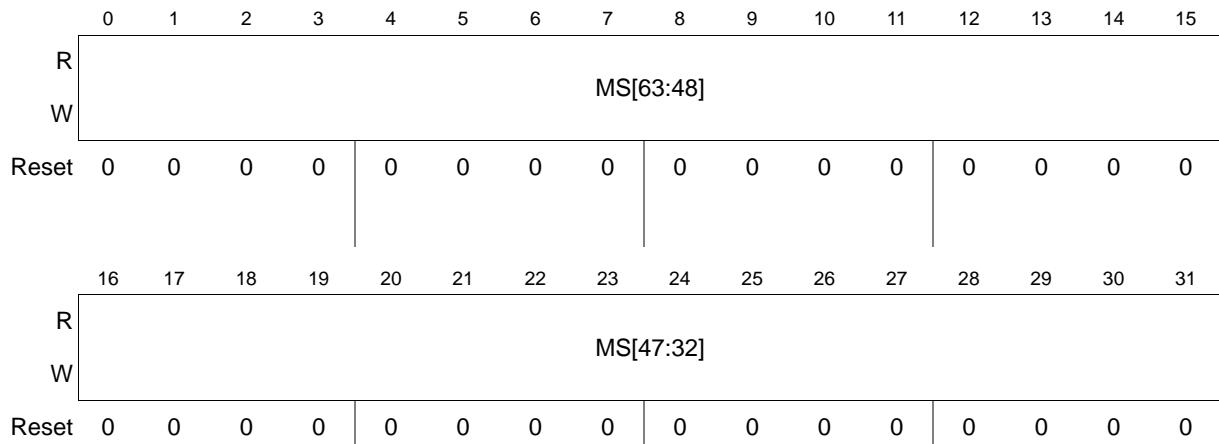


Figure 524. DFlash User Multiple Input Signature Register 1 (DFLASH\_UMISR1)

Table 487. DFLASH\_UMISR1 field descriptions

Field	Description
MS[63:32]	Multiple input Signature, bits 63-32 These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the flash memory. The MS can be seeded to any value by writing the DFLASH_UMISR1.

### 30.5.2.11 DFlash User Multiple Input Signature Register 2 (DFLASH\_UMISR2)

The DFLASH\_UMISR2 provides a mean to evaluate the Array Integrity.

The DFLASH\_UMISR2 represents the bits 95:64 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH\_UMISR2 is not accessible whenever DFLASH\_MCR[**DONE**] or DFLASH\_UT0[**AID**] are low: reading returns indeterminate data while writing has no effect.

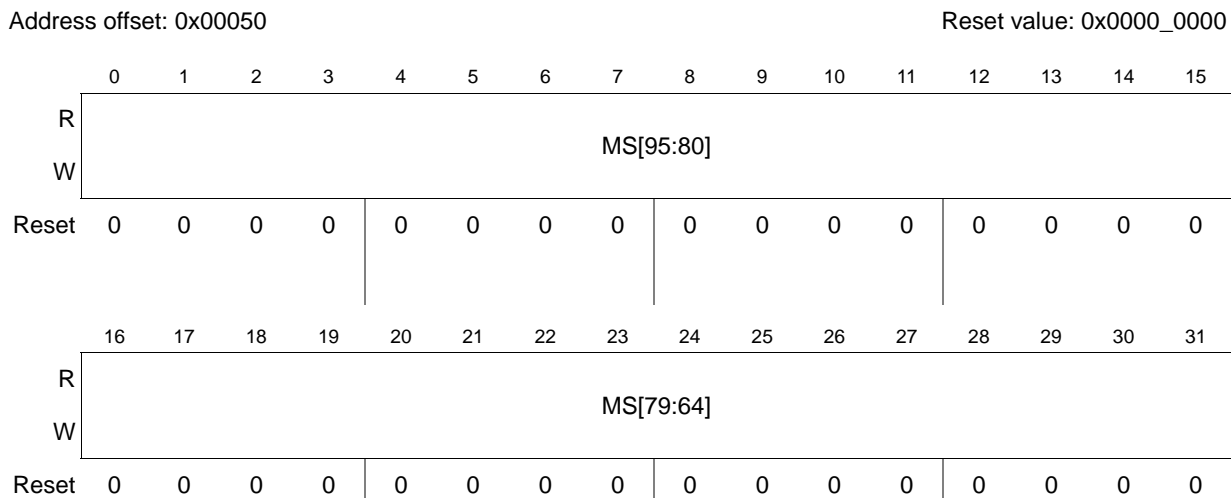


Figure 525. DFlash User Multiple Input Signature Register 2 (DFLASH\_UMISR2)

Table 488. DFLASH\_UMISR2 field descriptions

Field	Description
MS[95:64]	Multiple input Signature, bits 95-64 These bits represent the MISR value obtained accumulating the bits 95:64 of all the pages read from the flash memory. The MS can be seeded to any value by writing the DFLASH_UMISR2.

**30.5.2.12 DFlash User Multiple Input Signature Register 3 (DFLASH\_UMISR3)**

The DFLASH\_UMISR3 provides a mean to evaluate the Array Integrity.

The DFLASH\_UMISR3 represents the bits 127:96 of the whole 144 bits word (2 Double Words including ECC).

The DFLASH\_UMISR3 is not accessible whenever DFLASH\_MCR[**DONE**] or DFLASH\_UT0[**AID**] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x00054

Access: Read/write

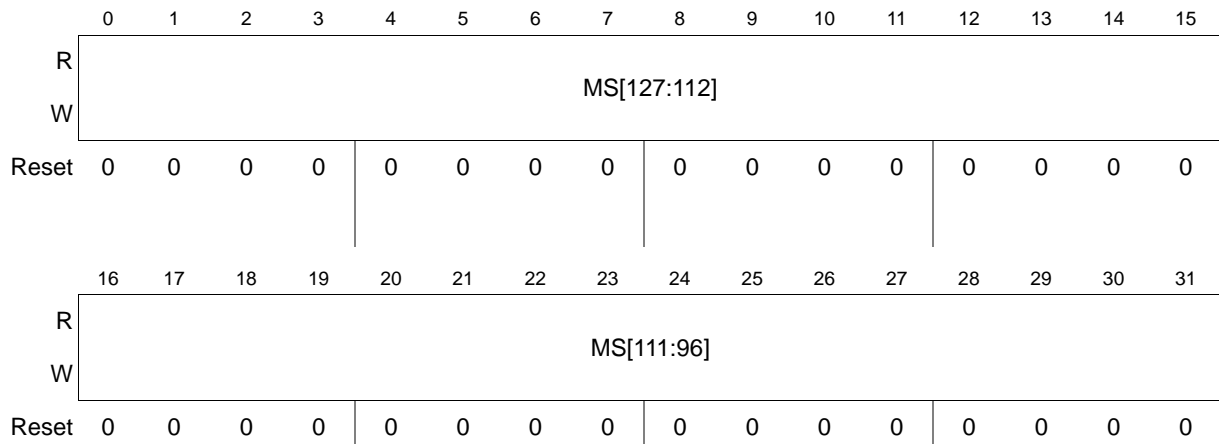


Figure 526. DFlash User Multiple Input Signature Register 3 (DFLASH\_UMISR3)

Table 489. DFLASH\_UMISR3 field descriptions

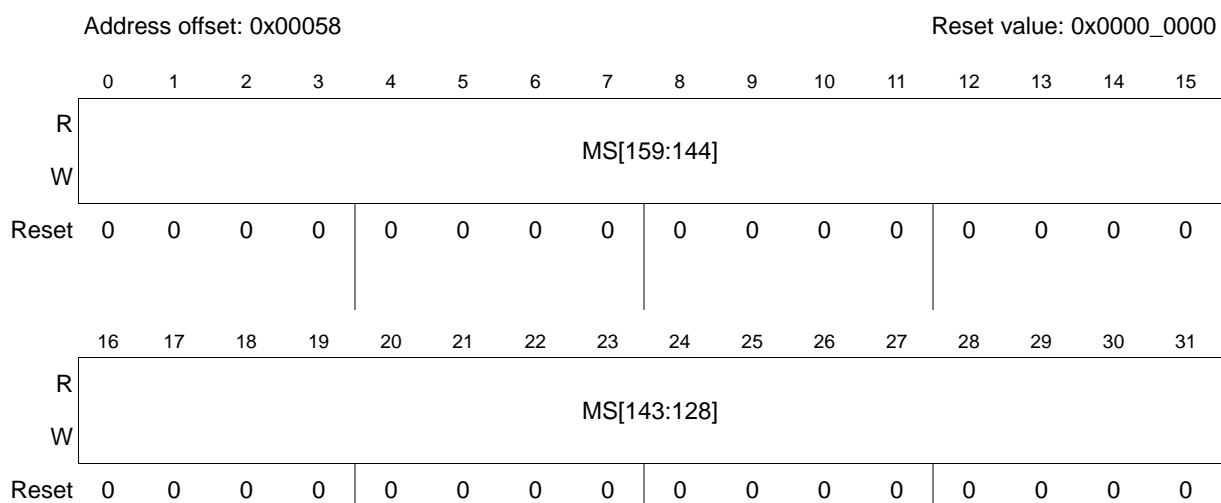
Field	Description
MS[127:96]	Multiple input Signature, bits 127:96 These bits represent the MISR value obtained accumulating the bits 127:96 of all the pages read from the flash memory. The MS can be seeded to any value by writing the DFLASH_UMISR3.

### 30.5.2.13 DFlash User Multiple Input Signature Register 4 (DFLASH\_UMISR4)

The Multiple Input Signature Register provides a mean to evaluate the Array Integrity.

The User Multiple Input Signature Register 4 represents the ECC bits of the whole 144 bits word (2 Double Words including ECC): bits 23-168:15 are ECC bits for the odd Double Word and bits 7-024:31 are the ECC bits for the even Double Word; bits 27-264:5 and 11-1020:21 of MISR are respectively the double and single ECC error detection for odd and even Double Word.

The DFLASH\_UMISR4 Register is not accessible whenever DFLASH\_MCR[*DONE*] or DFLASH\_UT0[*AID*] are low: reading returns indeterminate data while writing has no effect.



**Figure 527. DFlash User Multiple Input Signature Register 4 (DFLASH\_UMISR4)**

**Table 490. DFLASH\_UMISR4 field descriptions**

Field	Description
MS[159:128]	Multiple input Signature, bits 159-128 These bits represent the MISR value obtained accumulating: the 8 ECC bits for the even Double Word (on MS[135:128]); the single ECC error detection for even Double Word (on MS138); the double ECC error detection for even Double Word (on MS139); the 8 ECC bits for the odd Double Word (on MS[151:144]); the single ECC error detection for odd Double Word (on MS154); the double ECC error detection for odd Double Word (on MS155). The MS can be seeded to any value by writing the DFLASH_UMISR4 register.

## 30.6 Programming considerations

In the following sections, register names can refer to the CFlash or DFlash versions of those registers. Thus, for example, the term “MCR” can refer to the CFLASH\_MCR or DFLASH\_MCR based on context.

### 30.6.1 Modify operation

All modify operations of the flash memory module are managed through the flash memory User Registers Interface.

All the sectors of the flash memory module belong to the same partition (Bank), therefore when a Modify operation is active on some sectors no read access is possible on any other sector (Read-While-Write is not supported).

During a flash memory modify operation any attempt to read any flash memory location will output invalid data and bit MCR[RWE] will be automatically set. This means that the flash memory module is not fetchable when a modify operation is active and these commands must be executed from another memory (internal SRAM or another flash memory module).

If during a Modify Operation a reset occurs, the operation is suddenly terminated and the Macrocell is reset to Read Mode. The data integrity of the flash memory section where the

Modify Operation has been terminated is not guaranteed: the interrupted flash memory Modify Operation must be repeated.

In general each modify operation is started through a sequence of three steps:

1. The first instruction is used to select the desired operation by setting its corresponding selection bit in MCR (PGM or ERS) or UT0 (MRE or EIE).
2. The second step is the definition of the operands: the Address and the Data for programming or the Sectors for erase or margin read.
3. The third instruction is used to start the modify operation, by setting MCR[EHV] or UT0[AIE].

Once selected, but not yet started, one operation can be canceled by resetting the operation selection bit.

A summary of the available flash memory modify operations is shown in [Table 491](#).

**Table 491. Flash memory modify operations**

Operation	Select bit	Operands	Start bit
Double word program	MCR[PGM]	Address and data by interlock writes	MCR[EHV]
Sector erase	MCR[ERS]	LMS	MCR[EHV]
Array integrity check	None	LMS	UT0[AIE]
Margin read	UT0[MRE]	UT0[MRV] + LMS	UT0[AIE]
ECC Logic Check	UT0[EIE]	UT0[DSI], UT1, UT2	UT0[AIE]

Once the MCR[EHV] bit (or UT0[AIE]) is set, all the operands can no more be modified until the MCR[DONE] bit (or UT0[AID]) is high.

In general each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for the MCR[DONE] bit (or UT0[AID]) to go high.
2. Check operation result: check the MCR[PEG] bit (or compare UMISR0-4 with expected value).
3. Switch off FPEC by resetting the MCR[EHV] bit (or UT0[AIE]).
4. Deselect current operation by clearing the MCR[PGM] / MCR[ERS] fields (or UT0[MRE] / UT0[EIE]).

If the device embeds more than one flash memory module and a modify operation is on-going on one of them, then it is forbidden to start any other modify operation on the other flash memory modules.

In the following all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

### 30.6.2 Double word program

A flash memory Program sequence operates on any Double Word within the flash memory core.

Up to two words within the Double Word may be altered in a single Program operation.

ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed

since ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be of 64 bits. The programming operation should completely fill selected ECC segments within the Double Word.

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1.

Addresses in locked/disabled blocks cannot be programmed.

The user may program the values in any or all of two words, of a Double Word, with a single program sequence.

Double Word-bound words have addresses which differ only in address bit 2.

The Program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from 0 to 1.
2. Ensure the block that contains the address to be programmed is unlocked.  
Write the first address to be programmed with the program data.  
The flash memory module latches address bits (22:3) at this time.  
The flash memory module latches data written as well.  
This write is referred to as a program data interlock write. An interlock write may be as large as 64 bits, and as small as 32 bits (depending on the CPU bus).
3. If more than 1 word is to be programmed, write the additional address in the Double Word with data to be programmed. This is referred to as a program data write.  
The flash memory module ignores address bits (22:3) for program data writes.  
The eventual unwritten data word default to 0xFFFFFFFF.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm that the MCR[PEG] bit is 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program operation.

Program may be initiated with the 0 to 1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program.

The first write after a program is initiated determines the page address to be programmed. This first write is referred to as an interlock write. The interlock write determines if the shadow, test or normal array space will be programmed by causing the MCR[PEAS] field to be set/cleared.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

After the interlock write, additional writes only affect the data to be programmed at the word location determined by address bit 2. Unwritten locations default to a data value of 0xFFFFFFFF. If multiple writes are done to the same location the data for the last write is used in programming.

While MCR[DONE] is low and MCR[EHV] is high, the user may clear EHV, resulting in a program abort.

A Program abort forces the module to step 8 of the program sequence.

An aborted program will result in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed.

The data space being operated on before the abort will contain indeterminate data. This may be recovered by repeating the same program instruction or executing an erase of the affected blocks.

**Example 10** Double word program of data 0x55AA55AA at address 0x00AAA8 and data 0xAA55AA55 at address 0x00AAAC

```
MCR = 0x00000010; /* Set PGM in MCR: Select Operation */
(0x00AAA8) = 0x55AA55AA; /* Latch Address and 32 LSB data */
(0x00AAAC) = 0xAA55AA55; /* Latch 32 MSB data */
MCR = 0x00000011; /* Set EHV in MCR: Operation Start */
do
    /* Loop to wait for DONE=1 */
{ tmp = MCR; /* Read MCR */
} while ( !(tmp & 0x00000400) );
status = MCR & 0x00000200; /* Check PEG flag */
MCR = 0x00000010; /* Reset EHV in MCR: Operation End */
MCR = 0x00000000; /* Reset PGM in MCR: Deselect Operation */
```

### 30.6.3 Sector erase

Erase changes the value stored in all bits of the selected block(s) to logic 1.

An erase sequence operates on any combination of blocks (sectors) in the low, mid or high address space, or the shadow sector (if available). The test block cannot be erased.

The erase sequence is fully automated within the flash memory. The user only needs to select the blocks to be erased and initiate the erase sequence.

Locked/disabled blocks cannot be erased.

If multiple blocks are selected for erase during an erase sequence, no specific operation order must be assumed.

The erase operation consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to 1.
2. Select the block(s) to be erased by writing '1's to the appropriate bit(s) in the LMS register.  
If the shadow sector is to be erased, this step may be skipped, and LMS is ignored.  
Note that Lock and Select are independent. If a block is selected and locked, no erase will occur.
3. Write to any address in flash memory. This is referred to as an erase interlock write.
4. Write a logic 1 to the MCR[EHV] bit to start the internal erase sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase operation.



After setting MCR[ERS], one write, referred to as an interlock write, must be performed before MCR[EHV] can be set to '1'. Data words written during erase sequence interlock writes are ignored.

The user may terminate the erase sequence by clearing ERS before setting EHV.

An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low, MCR[EHV] is high and MCR[ESUS] is low.

An erase abort forces the module to step 8 of the erase sequence.

An aborted erase will result in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed.

The block(s) being operated on before the abort contain indeterminate data. This may be recovered by executing an erase on the affected blocks.

The user may not abort an erase sequence while in erase suspend.

#### Example 11 Erase of sectors B0F1 and B0F2

```
MCR = 0x00000004; /* Set ERS in MCR: Select Operation */
LMS = 0x00000006; /* Set LSL2-1 in LMS: Select Sectors to erase */
(0x000000) = 0xFFFFFFFF; /* Latch a flash memory Address with any data */
MCR = 0x00000005; /* Set EHV in MCR: Operation Start */
do /* Loop to wait for DONE=1 */
{ tmp = MCR; /* Read MCR */
} while ( !(tmp & 0x00000400) );
status = MCR & 0x00000200; /* Check PEG flag */
MCR = 0x00000004; /* Reset EHV in MCR: Operation End */
MCR = 0x00000000; /* Reset ERS in MCR: Deselect Operation */
```

### 30.6.3.1 Erase suspend/resume

The erase sequence may be suspended to allow read access to the flash memory core.

It is not possible to program or to erase during an erase suspend.

During erase suspend, all reads to blocks targeted for erase return indeterminate data.

An erase suspend can be initiated by changing the value of the MCR[ESUS] bit from 0 to 1. MCR[ESUS] can be set to '1' at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0 to 1 transition of MCR[ESUS] causes the module to start the sequence which places it in erase suspend.

The user must wait until MCR[DONE] = 1 before the module is suspended and further actions are attempted. MCR[DONE] will go high no more than  $t_{ESUS}$  after MCR[ESUS] is set to '1'.

Once suspended, the array may be read. flash memory core reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

#### Example 12 Sector erase suspend

```
MCR = 0x00000007; /* Set ESUS in MCR: Erase Suspend */
do /* Loop to wait for DONE=1 */
{ tmp = MCR; /* Read MCR */
} while ( !(tmp & 0x00000400) );
```

Notice that there is no need to clear MCR[EHV] and MCR[ERS] in order to perform reads during erase suspend.

The erase sequence is resumed by writing a logic 0 to MCR[ESUS].

MCR[EHV] must be set to '1' before MCR[ESUS] can be cleared to resume the operation.

The module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

**Example 13** Sector erase resume

```
MCR = 0x00000005; /* Reset ESUS in MCR: Erase Resume */
```

### 30.6.3.2 User Test mode

The user can perform specific tests to check flash memory module integrity by putting the flash memory module in User Test Mode.

Three kinds of test can be performed:

- Array Integrity Self Check
- Margin Read
- ECC Logic Check

The User Test Mode is equivalent to a Modify operation: read accesses attempted by the user during User Test Mode generates a Read-While-Write Error (MCR[RWE] set).

It is not allowed to perform User Test operations on the Test and shadow sectors.

#### 30.6.3.2.1 Array integrity self check

Array Integrity is checked using a predefined address sequence (proprietary), and this operation is executed on selected and unlocked blocks. Once the operation is completed, the results of the reads can be checked by reading the MISR value (stored in UMISR0–4), to determine if an incorrect read, or ECC detection was noted.

The internal MISR calculator is a 32-bit register.

The 128 bit data, the 16 ECC data and the single and double ECC errors of the two Double Words are therefore captured by the MISR through five different read accesses at the same location.

The whole check is done through five complete scans of the memory address space:

1. The first pass will scan only bits 31:0 of each page.
2. The second pass will scan only bits 63:32 of each page.
3. The third pass will scan only bits 95:64 of each page.
4. The fourth pass will scan only bits 127:96 of each page.
5. The fifth pass will scan only the ECC bits (8 + 8) and the single and double ECC errors (2 + 2) of both Double Words of each page.

The 128 bit data and the 16 ECC data are sampled before the eventual ECC correction, while the single and double error flags are sampled after the ECC evaluation.

Only data from existing and unlocked locations are captured by the MISR.

The MISR can be seeded to any value by writing the UMISR0–4 registers.

The Array Integrity Self Check consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing '1's to the appropriate bit(s) in the LMS register.  
Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.
3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Write a logic 1 to the UT0[AIE] bit to start the Array Integrity Check.
5. Wait until the UT0[AID] bit goes high.
6. Compare UMISR0-4 content with the expected result.
7. Write a logic 0 to the UT0[AIE] bit.
8. If more blocks are to be checked, return to step 2.

It is recommended to leave UT0[AIS] at 0 and use the proprietary address sequence that checks the read path more fully, although this sequence takes more time. During the execution of the Array Integrity Check operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way. While UT0[AID] is low and UT0[AIE] is high, the User may clear AIE, resulting in a Array Integrity Check abort.

UT0[AID] must be checked to know when the aborting command has completed.

**Example 14** Array integrity check of sectors B0F1 and B0F2

```

UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
LMS = 0x00000006; /* Set LSL2-1 in LMS: Select Sectors */
UT0 = 0x80000002; /* Set AIE in UT0: Operation Start */
do
    /* Loop to wait for AID=1 */
{ tmp= UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0= UMISR0; /* Read UMISR0 content*/
data1= UMISR1; /* Read UMISR1 content*/
data2= UMISR2; /* Read UMISR2 content*/
data3= UMISR3; /* Read UMISR3 content*/
data4= UMISR4; /* Read UMISR4 content*/
UT0 = 0x00000000; /* Reset UTE and AIE in UT0: Operation End */

```

### 30.6.3.2.2 Margin read

Margin read procedure (either Margin 0 or Margin 1), can be run on unlocked blocks in order to unbalance the Sense Amplifiers, respect to standard read conditions, so that all the read accesses reduce the margin vs '0' (UT0[MRV] = '0') or vs '1' (UT0[MRV] = '1'). Locked sectors are ignored by MISR calculation and ECC flagging. The results of the margin reads can be checked comparing checksum value in UMISR0-4. Since Margin reads are done at voltages that differ than the normal read voltage, lifetime expectancy of the flash memory macrocell is impacted by the execution of Margin reads. Doing Margin reads repetitively results in degradation of the flash memory Array, and shorten expected lifetime experienced at normal read levels. For these reasons the Margin Read usage is allowed only in Factory, while it is forbidden to use it inside the User Application.

In any case the charge losses detected through the Margin Read cannot be considered failures of the device and no Failure Analysis will be opened on them. The Margin Read Setup operation consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1's to the appropriate bit(s) in the LMS register.

Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set T0.AIS bit for a sequential addressing only.
4. Change the value in the UT0[MRE] bit from 0 to 1.
5. Select the Margin level: UT0[MRV]=0 for 0's margin, UT0[MRV]=1 for 1's margin.
6. Write a logic 1 to the UT0[AIE] bit to start the Margin Read Setup or skip to step 6 to terminate.
7. Wait until the UT0[AID] bit goes high.
8. Compare UMISR0-4 content with the expected result.
9. Write a logic 0 to the UT0[AIE], UT0[MRE] and UT0[MRV] bits.
10. If more blocks are to be checked, return to step 2.

It is mandatory to leave UT0[AIS] at 1 and use the linear address sequence, the usage of the proprietary sequence in Margin Read is forbidden.

During the execution of the Margin Read operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

The read accesses will be done with the addition of a proper number of Wait States to guarantee the correctness of the result.

While UT0[AID] is low and UT0[AIE] is high, the User may clear AIE, resulting in a Array Integrity Check abort.

UT0[AID] must be checked to know when the aborting command has completed.

#### Example 15 Margin read setup versus '1's

```

UMISR0 = 0x00000000; /* Reset UMISR0 content */
UMISR1 = 0x00000000; /* Reset UMISR1 content */
UMISR2 = 0x00000000; /* Reset UMISR2 content */
UMISR3 = 0x00000000; /* Reset UMISR3 content */
UMISR4 = 0x00000000; /* Reset UMISR4 content */

UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
LMS = 0x00000006; /* Set LSL2-1 in LMS: Select Sectors */
UT0 = 0x80000004; /* Set AIS in UT0: Select Operation */
UT0 = 0x80000024; /* Set MRE in UT0: Select Operation */
UT0 = 0x80000034; /* Set MRV in UT0: Select Margin versus 1's */
UT0 = 0x80000036; /* Set AIE in UT0: Operation Start */
do
    /* Loop to wait for AID=1 */
{ tmp = UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0 = UMISR0; /* Read UMISR0 content*/

```

```

data1 = UMISR1; /* Read UMISR1 content*/
data2 = UMISR2; /* Read UMISR2 content*/
data3 = UMISR3; /* Read UMISR3 content*/
data4 = UMISR4; /* Read UMISR4 content*/
UT0   = 0x80000034; /* Reset AIE in UT0: Operation End */
UT0   = 0x00000000; /* Reset UTE, MRE, MRV, AIS in UT0: Deselect Op. */

```

To exit from the Margin Read Mode a Read Reset operation must be executed.

### 30.6.3.2.3 ECC logic check

ECC logic can be checked by forcing the input of ECC logic: The 64 bits of data and the 8 bits of ECC syndrome can be individually forced and they will drive simultaneously at the same value the ECC logic of the whole page (2 Double Words).

The results of the ECC Logic Check can be verified by reading the MISR value.

The ECC Logic Check operation consists of the following sequence of events:

1. Set UT0[U<sub>TE</sub>] by writing the related password in UT0.
2. Write in UT1[D<sub>AI</sub>31–0] and UT2[D<sub>AI</sub>63–32] the Double Word Input value.
3. Write in UT0[D<sub>SI</sub>7–0] the Syndrome Input value.
4. Select the ECC Logic Check: write a logic 1 to the UT0[E<sub>IE</sub>] bit.
5. Write a logic 1 to the UT0[A<sub>IE</sub>] bit to start the ECC Logic Check.
6. Wait until the UT0[A<sub>ID</sub>] bit goes high.
7. Compare UMISR0–4 content with the expected result.
8. Write a logic 0 to the UT0[A<sub>IE</sub>] bit.

Notice that when UT0[A<sub>ID</sub>] is low UMISR0–4, UT1–2 and bits MRE, MRV, EIE, AIS and D<sub>SI</sub>7–0 of UT0 are not accessible: reading returns indeterminate data and write has no effect.

#### Example 16 ECC logic check

```

UT0   = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
UT1   = 0x55555555; /* Set DAI31-0 in UT1: Even Word Input Data */
UT2   = 0xAAAAAAAA; /* Set DAI63-32 in UT2: Odd Word Input Data */
UT0   = 0x80FF0000; /* Set DSI7-0 in UT0: Syndrome Input Data */
UT0   = 0x80FF0008; /* Set EIE in UT0: Select ECC Logic Check */
UT0   = 0x80FF000A; /* Set AIE in UT0: Operation Start */
do
    /* Loop to wait for AID=1 */
{ tmp = UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0 = UMISR0; /* Read UMISR0 content (expected 0x55555555) */
data1 = UMISR1; /* Read UMISR1 content (expected 0xAAAAAAAA) */
data2 = UMISR2; /* Read UMISR2 content (expected 0x55555555) */
data3 = UMISR3; /* Read UMISR3 content (expected 0xAAAAAAAA) */
data4 = UMISR4; /* Read UMISR4 content (expected 0x00FF00FF) */
UT0   = 0x00000000; /* Reset UTE, AIE and EIE in UT0: Operation End */

```

### 30.6.3.3 Error correction code

The flash memory module provides a method to improve the reliability of the data stored in flash memory: the usage of an Error Correction Code. The word size is fixed at 64 bits.

Eight ECC bits, programmed to guarantee a Single Error Correction and a Double Error Detection (SEC-DED), are associated to each 64-bit Double Word.

ECC circuitry provides correction of single bit faults and is used to achieve automotive reliability targets. Some units will experience single bit corrections throughout the life of the product with no impact to product reliability.

#### 30.6.3.3.1 ECC algorithms

The flash memory module supports one ECC Algorithm: "All '1's No Error". A modified Hamming code is used that ensures the all erased state (that is, 0xFFFF....FFFF) data is a valid state, and will not cause an ECC error. This allows the user to perform a blank check after a sector erase operation.

#### 30.6.3.4 EEPROM emulation

The chosen ECC algorithm allows some bit manipulations so that a Double Word can be rewritten several times without needing an erase of the sector. This allows to use a Double Word to store flags useful for the Eeprom Emulation. As an example the chosen ECC algorithm allows to start from an All '1's Double Word value and rewrite whichever of its four 16-bits Half-Words to an All '0's content by keeping the same ECC value.

[Table 492](#) shows a set of Double Words sharing the same ECC value.

**Table 492. Bit manipulation: Double words with the same ECC value**

Double word	ECC all '1's no error
0xFFFF_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_FFFF_0000	0xFF
0xFFFF_FFFF_0000_FFFF	0xFF
0xFFFF_0000_FFFF_FFFF	0xFF
0x0000_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_0000_0000	0xFF
0xFFFF_0000_FFFF_0000	0xFF
0x0000_FFFF_FFFF_0000	0xFF
0xFFFF_0000_0000_FFFF	0xFF
0x0000_FFFF_0000_FFFF	0xFF
0x0000_0000_FFFF_FFFF	0xFF
0xFFFF_0000_0000_0000	0xFF
0x0000_FFFF_0000_0000	0xFF
0x0000_0000_0000_0000	0xFF

When some flash memory sectors are used to perform an Eeprom Emulation, it is recommended for safety reasons to reserve at least 3 sectors to this purpose.

#### 30.6.3.4.1 All '1's No Error

The All '1's No Error Algorithm detects as valid any Double Word read on a just erased sector (all the 72 bits are '1's).

This option allows to perform a Blank Check after a Sector Erase operation.

#### 30.6.3.5 Protection strategy

Two kinds of protection are available: Modify Protection to avoid unwanted program/erase in flash memory sectors and Censored Mode to avoid piracy.

##### 30.6.3.5.1 Modify protection

The flash memory Modify Protection information is stored in nonvolatile flash memory cells located in the TestFlash. This information is read once during the flash memory initialization phase following the exiting from Reset and is stored in volatile registers that act as actuators.

The reset state of all the volatile modify protection registers is the protected state.

All the nonvolatile modify protection registers can be programmed through a normal Double Word Program operation at the related locations in TestFlash.

The nonvolatile modify protection registers cannot be erased.

- The nonvolatile Modify Protection Registers are physically located in TestFlash their bits can be programmed to '0' only once and they can no more be restored to '1'.
- The Volatile Modify Protection Registers are Read/Write registers which bits can be written at '0' or '1' by the user application.

A software mechanism is provided to independently lock/unlock each Low, Mid and High Address Space Block against program and erase.

Software locking is done through the LML register.

An alternate means to enable software locking for blocks of Low Address Space only is through the SLL.

All these registers have a nonvolatile image stored in TestFlash (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery the TestFlash nonvolatile image is at all '1's, meaning all sectors are locked.

By programming the nonvolatile locations in TestFlash the selected sectors can be unlocked.

Being the TestFlash One Time Programmable (that is, not erasable), once unlocked the sectors cannot be locked again.

Of course, on the contrary, all the volatile registers can be written at 0 or 1 at any time, therefore the user application can lock and unlock sectors when desired.

##### 30.6.3.5.2 Censored mode

The Censored Mode information is stored in nonvolatile flash memory cells located in the Shadow Sector. This information is read once during the flash memory initialization phase following the exiting from Reset and is stored in volatile registers that act as actuators.

The reset state of all the Volatile Censored Mode Registers is the protected state.

All the nonvolatile Censored Mode registers can be programmed through a normal Double Word Program operation at the related locations in the Shadow Sector.

The nonvolatile Censored Mode registers can be erased by erasing the Shadow Sector.

- The nonvolatile Censored Mode Registers are physically located in the Shadow Sector their bits can be programmed to '0' and restored to '1' by erasing the Shadow Sector.
- The Volatile Censored Mode Registers are registers not accessible by the user application.

The flash memory module provides two levels of protection against piracy:

- If bits CW15:0 of NVSCC0 are programmed at 0x55AA and NVSC1 = NVSCC0 the Censored Mode is disabled, while all the other possible values enable the Censored Mode.
- If bits SC15:0 of NVSCC0 are programmed at 0x55AA and NVSC1 = NVSCC0 the Public Access is disabled, while all the other possible values enable the Public Access.

The parts are delivered to the user with Censored Mode and Public Access disabled.

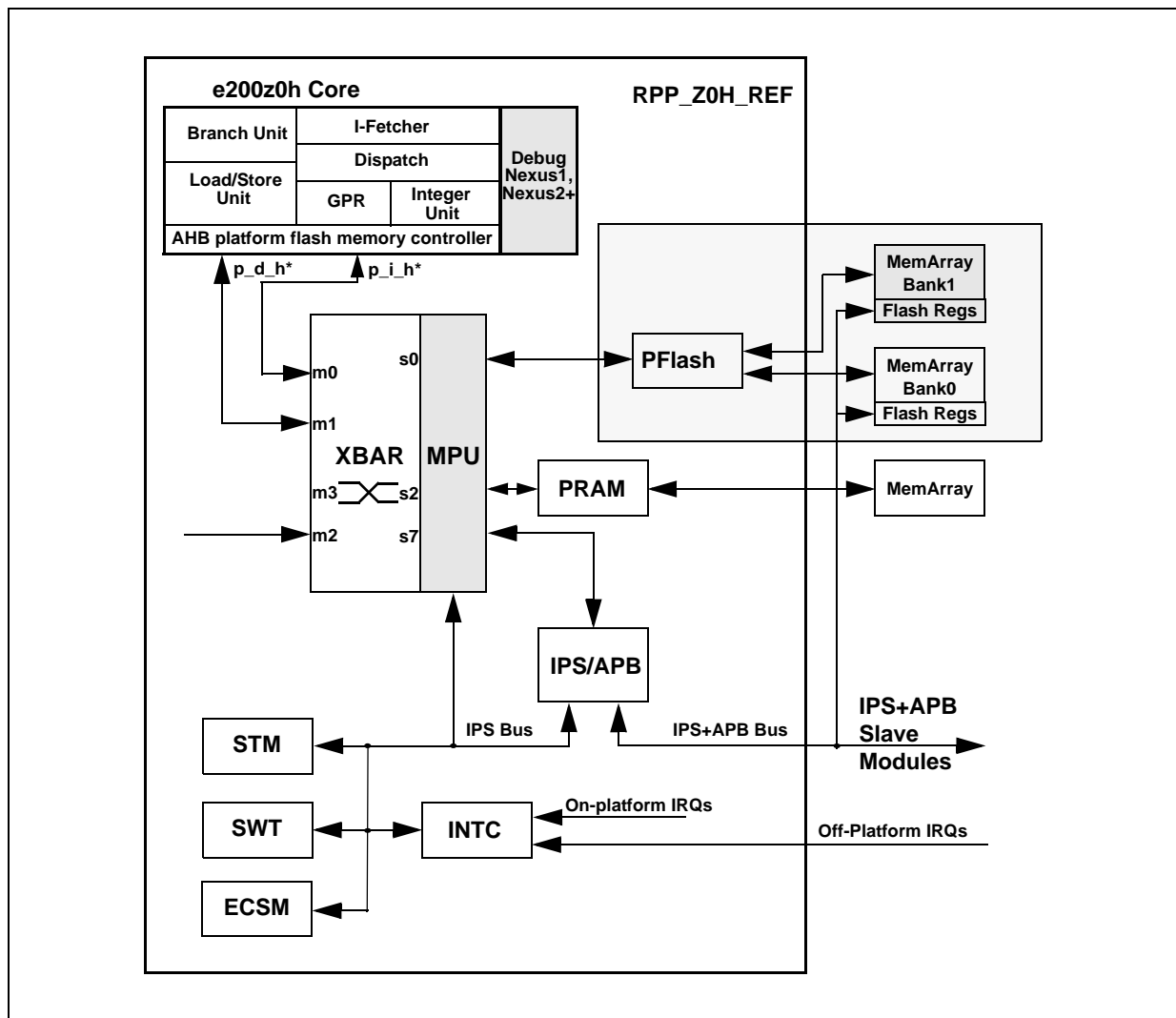
## 30.7 Platform flash memory controller

### 30.7.1 Introduction

The platform flash memory controller acts as the interface between the system bus (AHB-Lite 2.v6) and up to two banks of integrated flash memory arrays (Program and Data). It intelligently converts the protocols between the system bus and the dedicated flash memory array interfaces.

A block diagram of the e200z0h Power Architecture reduced product platform (RPP) reference design is shown below in [Figure 528](#) with the platform flash memory controller module and its attached off-platform flash memory arrays highlighted.





**Figure 528. Power Architecture e200z0h RPP reference platform block diagram**

The module list includes:

- Power Architecture e200z0h (Harvard) core with Nexus1 or optional Nexus2+ debug
- AHB crossbar switch "lite" (XBAR)
- Memory Protection Unit (MPU)
- Platform flash memory controller with connections to 2 memory banks
- Platform SRAM memory controller (PRAM)
- AHB-to-IPS/APB bus controller (PBRIDGE) for access to on- and off-platform slave modules
- Interrupt Controller (INTC)
- 4-channel System Timers (STM)
- Software Watchdog Timer (SWT)
- Error Correction Status Module (ECSM)

Throughout this document, several important terms are used to describe the platform flash memory controller module and its connections. These terms are defined here:

- **Port** — This is used to describe the AMBA-AHB connection(s) into the platform flash memory controller. From an architectural and programming model viewpoint, the definition supports up to two AHB ports, even though this specific controller only supports a single AHB connection.
- **Bank** — This term is used to describe the attached flash memories. From the platform flash memory controller's perspective, there may be one or two attached banks of flash memory. The "code flash memory" is required and always attached to bank0. Additionally, there is a "data flash memory" attached to bank1. The platform flash memory controller interface supports two separate connections, one to each memory bank.
- **Array** — Within each memory bank, there is one flash memory array instantiations.
- **Page** — This value defines the number of bits read from the flash memory array in a single access. For this controller and memory, the page size is 128 bits (16 bytes).

The nomenclature "page buffers" and "line buffers" are used interchangeably.

### 30.7.1.1 Overview

The platform flash memory controller supports a 32-bit data bus width at the AHB port and connections to 128-bit read data interfaces from two memory banks, where each bank contains one instantiations of the flash memory array. One flash memory bank is connected to the code flash memory and the other bank is connected to the optional data flash memory. The memory controller capabilities vary between the two banks with each bank's functionality optimized with the typical use cases associated with the attached flash memory. As an example, the platform flash memory controller logic associated with the code flash memory bank contains a four-entry "page" buffer, each entry containing 128 bits of data (1 flash memory page) plus an associated controller which prefetches sequential lines of data from the flash memory array into the buffer, while the controller logic associated with the data flash memory bank only supports a 128-bit register which serves as a temporary page holding register and does not support any prefetching. Prefetch buffer hits from the code flash memory bank support zero-wait AHB data phase responses. AHB read requests which miss the buffers generate the needed flash memory array access and are forwarded to the AHB upon completion, typically incurring two wait-states at an operating frequency of 60–64 MHz.

This memory controller is optimized for applications where a cacheless processor core, e.g., the Power e200z0h, is connected through the platform to on-chip memories, e.g., flash memory and SRAM, where the processor and platform operate at the same frequency. For these applications, the 2-stage pipeline AMBA-AHB system bus is effectively mapped directly into stages of the processor's pipeline and zero wait-state responses for most memory accesses are critical for providing the required level of system performance.

### 30.7.1.2 Features

The following list summarizes the key features of the platform flash memory controller:

- Dual array interfaces support up to a total of 16 MB of flash memory, partitioned as two separate 8 MB banks
- Single AHB port interface supports a 32-bit data bus. All AHB aligned and unaligned reads within the 32-bit container are supported. Only aligned word writes are supported.
- Array interfaces support a 128-bit read data bus and a 64-bit write data bus for each bank
- Interface with code flash memory (bank0) provides configurable read buffering and page prefetch support. Four page read buffers (each 128 bits wide) and a prefetch controller are used to support single-cycle read responses (zero AHB data phase wait-states) for hits in the buffers. The buffers implement a least-recently-used replacement algorithm to maximize performance.
- Interface with optional data flash memory (bank1) includes a 128-bit register to temporarily hold a single flash memory page. This logic supports single-cycle read responses (zero AHB data phase wait-states) for accesses that hit in the holding register. There is no support for prefetching associated with this bank.
- Programmable response for read-while-write sequences including support for stall-while-write, optional stall notification interrupt, optional flash memory operation abort, and optional abort notification interrupt
- Separate and independent configurable access timing (on a per bank basis) to support use across a wide range of platforms and frequencies
- Support of address-based read access timing for emulation of other memory types
- Support for reporting of single- and multi-bit flash memory ECC events
- Typical operating configuration loaded into programming model by system reset

### 30.7.2 Memory map and register description

Two memory maps are associated with the platform flash memory controller: one for the flash memory space and another for the program-visible control and configuration registers. The flash memory space is accessed via the AMBA-AHB port and the program-visible registers are accessed via the slave peripheral bus. Details on both memory spaces are provided in [Section 30.7.2.1: Memory map](#).

There are no program-visible registers that physically reside inside the platform flash memory controller. Rather, the platform flash memory controller receives control and configuration information from the flash memory array controller(s) to determine the operating configuration. These are part of the flash memory array's configuration registers mapped into its slave peripheral (IPS) address space but are described here.

*Note: Updating the configuration fields that control the platform flash controller behavior should only occur while the flash controller is idle. Changing configuration settings while a flash access is in progress can lead to non-deterministic behavior.*

#### 30.7.2.1 Memory map

First, consider the flash memory space accessed via transactions from the platform flash memory controller's AHB port.

To support the two separate flash memory banks, each up to 8 MB in size, the platform flash memory controller uses address bit 23 (haddr[23]) to steer the access to the appropriate memory bank. In addition to the actual flash memory regions, the system memory map includes shadow and test sectors. The program-visible control and configuration registers associated with each memory array are included in the slave peripheral address region. The system memory map defines one code flash memory array and one data flash memory array. See [Table 493](#).

**Table 493. Flash memory-related regions in the system memory map**

Start address	End address	Size [KB]	Region
0x0008_0000	0x000F_FFFF	512	Code flash memory array 1
0x0010_0000	0x0017_FFFF	512	Code flash memory array 2
0x0018_0000	0x001F_FFFF	512	Reserved
0x0020_0000	0x0027_FFFF	16	Code flash memory array 0: shadow sector
0x0028_0000	0x002F_FFFF	1536	Reserved
0x0040_0000	0x0040_3FFF	16	Code flash memory array 0: test sector
0x0040_4000	0x007F_FFFF	4078	Reserved
0x0080_0000	0x0080_FFFF	64	Data flash memory array 0
0x0081_0000	0x00BF_FFFF	4032	Reserved
0x00C0_0000	0x00C7_FFFF	16	Data flash memory array 0: test sector
0x00C8_0000	0x00FF_FFFF	3584	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Emulation mapping
0xC3F8_8000	0xC3F8_BFFF	16	Code flash memory array 0 configuration
0xC3F8_C000	0xC3F8_FFFF	16	Data flash memory array 0 configuration

For additional information on the address-based read access timing for emulation of other memory types, see [Section 30.8.11: Wait-state emulation](#).

Next, consider the memory map associated with the control and configuration registers.

Regardless of the number of populated banks or the number of flash memory arrays included in a given bank, the configuration of the platform flash memory controller is wholly specified by the platform flash memory controller registers associated with code flash memory array 0. The code array0 register settings define the operating behavior of **both** flash memory banks; it is recommended that the platform flash memory controller registers for all physically-present arrays be set to the code flash memory array0 values.

*Note:* To perform program and erase operations, the control registers in the actual referenced flash memory array must be programmed, but the configuration of the platform flash memory controller module is defined by the platform flash controller registers of code array0.

The 32-bit memory map for the platform flash memory controller control registers is shown in [Table 494](#). The base address of the controller is 0xC3F8\_8000.

**Table 494. Platform flash memory controller 32-bit memory map**

Address offset	Register	Location
0x1C	Platform Flash Configuration Register 0 (PFCR0)	<a href="#">on page 909</a>
0x20	Platform Flash Configuration Register 1 (PFCR1)	<a href="#">on page 912</a>
0x24	Platform Flash Access Protection Register (PFAPR)	<a href="#">on page 914</a>

See the SPC560B54/6x data sheet for detailed settings for different values of frequency.

### 30.7.2.2 Register description

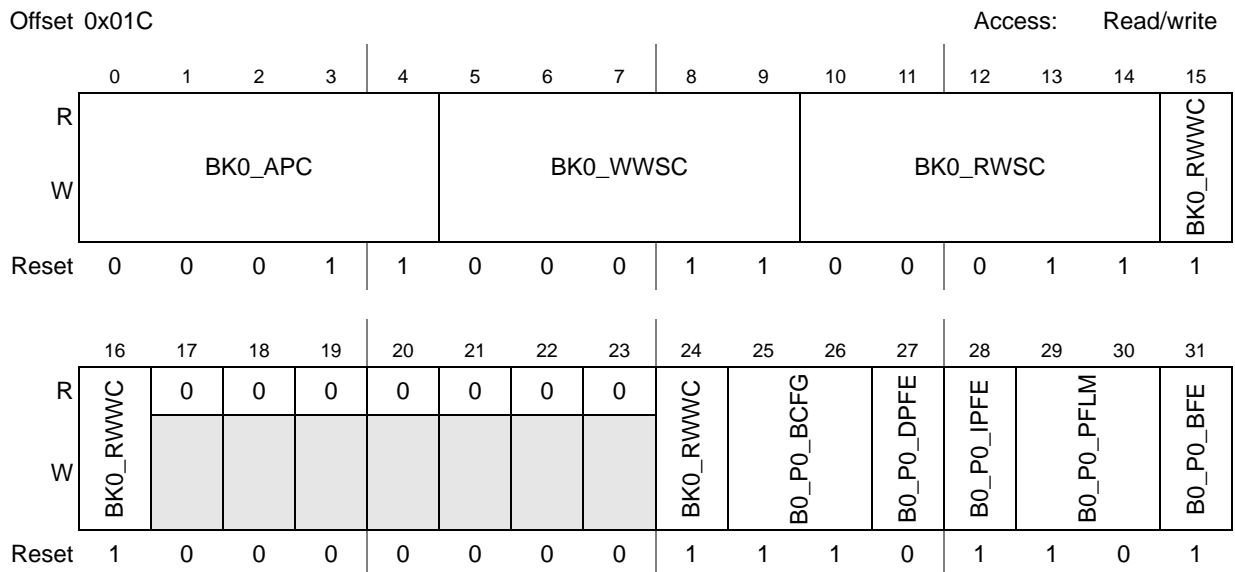
This section details the individual registers of the platform flash memory controller.

Flash memory configuration registers must be written only with 32-bit write operations to avoid any issues associated with register “incoherency” caused by bits spanning smaller-size (8- or 16-bit) boundaries.

#### 30.7.2.2.1 Platform Flash Configuration Register 0 (PFCR0)

This register defines the configuration associated with the code flash memory bank0. It includes fields that provide specific information for up to two separate AHB ports (p0 and the optional p1). For the platform flash memory controller module, the fields associated with AHB port p1 are ignored. The register is described in [Figure 529](#) and [Table 495](#).

*Note: Do not execute code from flash memory when you are programming PFCR0. If you wish to program PFCR0, execute your application code from RAM.*



**Figure 529. PFlash Configuration Register 0 (PFCR0)**

Table 495. PFCR0 field descriptions

Field	Description
BK0_APC	Bank0 Address Pipelining Control The setting for APC must be the same as RWSC.
BK0_WWSC	Bank0 Write Wait-State Control This field is used to control the number of wait-states to be added to the flash memory array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device datasheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. This field is set to an appropriate value by hardware reset.  00000: No additional wait-states are added 00001: One additional wait-state is added 00010: Two additional wait-states are added ... 11111: 31 additional wait-states are added <b>Note:</b> The Platform Flash Memory Controller does not support Write Wait-State Control since this capability is not supported by the flash memory array.
BK0_RWSC	Bank0 Read Wait-State Control This field is used to control the number of wait-states to be added to the flash memory array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. The required settings are documented in the device datasheet.  00000: No additional wait-states are added 00001: One additional wait-state is added 00010: Two additional wait-states are added ... 11111: 31 additional wait-states are added <b>Note:</b> The setting for RWSC must be the same as APC.
BK0_RWWC	Bank0 Read-While-Write Control This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation.  0—: This state should be avoided. Setting to this state can cause unpredictable operation. 111: Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt 110: Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt 101: Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt 100: Generate a bus stall for a read while write/erase, enable the operation abort and the abort notification interrupt  This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.

**Table 495. PFCR0 field descriptions (continued)**

Field	Description
B0_P0_BCFG	<p>Bank0, Port 0 Page Buffer Configuration</p> <p>This field controls the configuration of the four page buffers in the PFlash controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers.</p> <p>If enabled, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the flash memory access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit.</p> <p>00: All four buffers are available for any flash memory access, that is, there is no partitioning of the buffers based on the access type.                      01: Reserved                      10: The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses.                      11: The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses.</p> <p>This field is set to 2b11 by hardware reset.</p>
B0_P0_DPFE	<p>Bank0, Port 0 Data Prefetch Enable</p> <p>This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. Prefetching can be enabled/disabled on a per Master basis at PFAPR[MxPFD].</p> <p>0: No prefetching is triggered by a data read access                      1: If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any data read access</p>
B0_P0_IPFE	<p>Bank0, Port 0 Instruction Prefetch Enable</p> <p>This field enables or disables prefetching initiated by an instruction fetch read access. This field is set by hardware reset. Prefetching can be enabled/disabled on a per Master basis at PFAPR[MxPFD].</p> <p>0: No prefetching is triggered by an instruction fetch read access                      1: If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any instruction fetch read access</p>

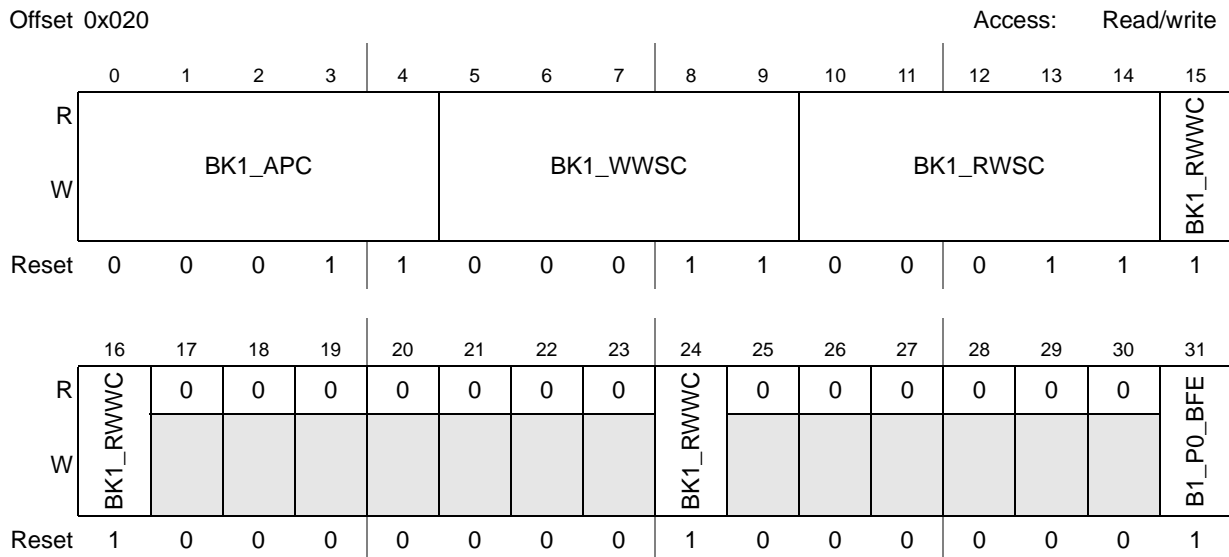
**Table 495. PFCR0 field descriptions (continued)**

Field	Description
B0_P0_PFLM	<p>Bank0, Port 0 Prefetch Limit</p> <p>This field controls the prefetch algorithm used by the PFlash controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is set to 2b10 by hardware reset.</p> <p>00: No prefetching is performed.                      01: The referenced line is prefetched on a buffer miss, that is, <i>prefetch on miss</i>.                      1–: The referenced line is prefetched on a buffer miss, or the next sequential page is prefetched on a buffer hit (if not already present), that is, <i>prefetch on miss or hit</i>.</p>
B0_P0_BFE	<p>Bank0, Port 0 Buffer Enable</p> <p>This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. This bit is set by hardware reset.</p> <p>0: The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared.                      1: The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.</p>

**30.7.2.2.2 Platform Flash Configuration Register 1 (PFCR1)**

This register defines the configuration associated with flash memory bank1. This corresponds to the “data flash memory”. It includes fields that provide specific information for up to two separate AHB ports (p0 and the optional p1). For the platform flash memory controller module, the fields associated with AHB port p1 are ignored. The register is described below in [Figure 530](#) and [Table 496](#).

*Note: Do not execute code from flash memory when you are programming PFCR1. If you wish to program PFCR1, execute your application code from RAM.*



**Figure 530. PFlash Configuration Register 1 (PFCR1)**



**Table 496. PFCR1 field descriptions**

Field	Description
BK1_APC	<p>Bank1 Address Pipelining Control                      This field is used to control the number of cycles between flash memory array access requests. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device datasheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation.</p> <p>00000: Accesses may be initiated on consecutive (back-to-back) cycles                      00001: Access requests require one additional hold cycle                      00010: Access requests require two additional hold cycles                      ...                      11110: Access requests require 30 additional hold cycles                      11111: Access requests require 31 additional hold cycles</p> <p>This field is ignored in single bank flash memory configurations.  <b>Note:</b> The Platform Flash Memory Controller does not support Address Pipelining Control since this capability is not supported by the flash memory array.</p>
BK1_WWSC	<p>Bank1 Write Wait-State Control                      This field is used to control the number of wait-states to be added to the flash memory array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device datasheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. This field is set to an appropriate value by hardware reset.</p> <p>00000: No additional wait-states are added                      00001: One additional wait-state is added                      00010: Two additional wait-states are added                      ...                      11111: 31 additional wait-states are added</p> <p>This field is ignored in single bank flash memory configurations.  <b>Note:</b> The Platform Flash Memory Controller does not support Write Wait-State Control since this capability is not supported by the flash memory array.</p>
BK1_RWSC	<p>Bank1 Read Wait-State Control                      This field is used to control the number of wait-states to be added to the flash memory array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. The required settings are documented in the device datasheet.</p> <p>00000: No additional wait-states are added                      00001: One additional wait-state is added                      00010: Two additional wait-states are added                      ...                      11111: 31 additional wait-states are added</p> <p>This field is ignored in single bank flash memory configurations.</p>

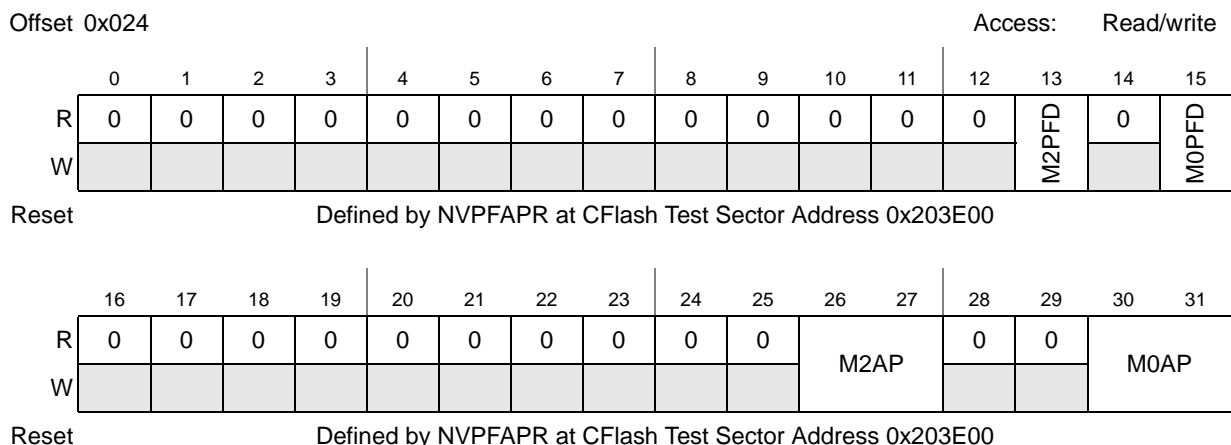
Table 496. PFCR1 field descriptions (continued)

Field	Description
BK1_RWWC	<p>Bank1 Read-While-Write Control This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation.</p> <p>0—: Terminate any attempted read while write/erase with an error response 111: Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt 110: Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt 101: Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt 100: Generate a bus stall for a read while write/erase, enable the operation abort and the abort notification interrupt</p> <p>This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.</p> <p>This field is ignored in single bank flash memory configurations.</p>
B1_P0_PFE	<p>Bank1, Port 0 Buffer Enable This bit enables or disables read hits from the 128-bit holding register. It is also used to invalidate the contents of the holding register. This bit is set by hardware reset, enabling the use of the holding register.</p> <p>0: The holding register is disabled from satisfying read requests. 1: The holding register is enabled to satisfy read requests on hits.</p>

### 30.7.2.2.3 Platform Flash Access Protection Register (PFAPR)

The PFlash Access Protection Register (PFAPR) is used to control read and write accesses to the flash memory based on system master number. Prefetching capabilities are defined on a per master basis. This register also defines the arbitration mode for controllers supporting two AHB ports. The register is described below in [Figure 531](#) and [Table 497](#).

The contents of the register are loaded from location 0x203E00 of the shadow region in the code flash memory (bank0) array at reset. To temporarily change the values of any of the fields in the PFAPR, a write to the IPS-mapped register is performed. To change the values loaded into the PFAPR *at reset*, the word location at address 0x203E00 of the shadow region in the flash memory array must be programmed using the normal sequence of operations. The reset value shown in [Table 531](#) reflects an erased or unprogrammed value from the shadow region.



**Figure 531. PFlash Access Protection Register (PFAPR)**

**Table 497. PFAPR field descriptions**

Field	Description
M2PFD	<p>eDMA Master 2 Prefetch Disable</p> <p>This field controls whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCR0[B0_Px_DPFE, B0_Px_IPFE, Bx_Py_BFE] bits. For master numbering, see <a href="#">Table 157</a>.</p> <p>0: Prefetching may be triggered by this master                      1: No prefetching may be triggered by this master</p>
M0PFD	<p>e200z0 core Master 0 Prefetch Disable</p> <p>This field controls whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCR0[B0_Px_DPFE, B0_Px_IPFE, Bx_Py_BFE] bits. For master numbering, see <a href="#">Table 157</a>.</p> <p>0: Prefetching may be triggered by this master                      1: No prefetching may be triggered by this master</p>
M2AP	<p>eDMA Master 2 Access Protection</p> <p>These fields control whether read and write accesses to the flash memory are allowed based on the master number of the initiating module. For master numbering, see <a href="#">Table 157</a>.</p> <p>00: No accesses may be performed by this master                      01: Only read accesses may be performed by this master                      10: Only write accesses may be performed by this master                      11: Both read and write accesses may be performed by this master</p>
M0AP	<p>e200z0 core Master 0 Access Protection</p> <p>These fields control whether read and write accesses to the flash memory are allowed based on the master number of the initiating module. For master numbering, see <a href="#">Table 157</a>.</p> <p>00: No accesses may be performed by this master                      01: Only read accesses may be performed by this master                      10: Only write accesses may be performed by this master                      11: Both read and write accesses may be performed by this master</p>

### 30.7.2.2.3.1 Nonvolatile Platform Flash Access Protection Register (NVPFAPR)

The NVPFAPR register has a related Nonvolatile PFAPR located in the Shadow Sector that contains the default reset value for PFAPR. During the reset phase of the flash memory module, the NVPFAPR register content is read and loaded into the PFAPR.

The NVPFAPR register is a 64-bit register, of which the 32 most significant bits 63:32 are 'don't care' and are used to manage ECC codes.

Offset: 0x203E00

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	1	1	1	1	1	1	1	1	1	1	1	1	M2PFD	1	M0PFD
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	M2AP		0	0	M0AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 532. Nonvolatile Platform Flash Access Protection Register (NVPFAPR)

Table 498. NVPFAPR field descriptions

Field	Description
M2PFD	See <a href="#">Table 497</a> .
M0PFD	See <a href="#">Table 497</a> .
M2AP	See <a href="#">Table 497</a> .
M0AP	See <a href="#">Table 497</a> .

## 30.8 Functional description

The platform flash memory controller interfaces between the AHB system bus and the flash memory arrays.

The platform flash memory controller generates read and write enables, the flash memory array address, write size, and write data as inputs to the flash memory array. The platform flash memory controller captures read data from the flash memory array interface and drives it onto the AHB. Up to four pages of data (128-bit width) from bank0 are buffered by the platform flash memory controller. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle (zero AHB wait-states) read data responses on buffer hits.

Several prefetch control algorithms are available for controlling page read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-master basis.

Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch; see [Section 30.7.2.2.1: Platform Flash Configuration Register 0 \(PFCR0\)](#), and [Section 30.7.2.2.2: Platform Flash Configuration Register 1 \(PFCR1\)](#).

Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms; see [Section 30.7.2.2.3: Platform Flash Access Protection Register \(PFAPR\)](#).

Throughout this discussion, `bkn_` is used as a prefix to refer to two signals, each for each bank: `bk0_` and `bk1_`. Also, the nomenclature `Bx_Py_RegName` is used to reference a program-visible register field associated with bank “x” and port “y”.

### 30.8.1 Access protections

The platform flash memory controller provides programmable configurable access protections for both read and write cycles from masters via the PFlash Access Protection Register (PFAPR). It allows restriction of read and write requests on a per-master basis. This functionality is described in [Section 30.7.2.2.3: Platform Flash Access Protection Register \(PFAPR\)](#). Detection of a protection violation results in an error response from the platform flash memory controller on the AHB transfer.

### 30.8.2 Read cycles – Buffer miss

Read cycles from the flash memory array are initiated by the platform flash memory controller. The platform flash memory controller then waits for the programmed number of read wait-states before sampling the read data from the flash memory array. This data is normally stored in the least-recently updated page read buffer for bank0 in parallel with the requested data being forwarded to the AHB. For bank1, the data is captured in the page-wide temporary holding register as the requested data is forwarded to the AHB bus.

If the flash memory access was the direct result of an AHB transaction, the page buffer is marked as most-recently-used as it is being loaded. If the flash memory access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least-recently-used buffer. The status of this buffer is not changed to most-recently-used until a subsequent buffer hit occurs.

### 30.8.3 Read cycles – Buffer hit

Single cycle read responses to the AHB are possible with the platform flash memory controller when the requested read access was previously loaded into one of the bank0 page buffers. In these “buffer hit” cases, read data is returned to the AHB data phase with a zero wait-state response.

Likewise, the bank1 logic includes a single 128-bit temporary holding register and sequential accesses which “hit” in this register are also serviced with a zero wait-state response.

### 30.8.4 Write cycles

Write cycles are initiated by the platform flash memory controller. The platform flash memory controller then waits for the appropriate number of write wait-states before terminating the write operation.

### 30.8.5 Error termination

The first case that can cause an error response to the AHB is when an access is attempted by an AHB master whose corresponding Read Access Control or Write Access Control settings do not allow the access, thus causing a protection violation. In this case, the platform flash memory controller does not initiate a flash memory array access.

The second case that can cause an error response to the AHB is when an access is performed to the flash memory array and is terminated with a flash memory error response. See [Section 30.8.7: Flash error response operation](#). This may occur for either a read or a write operation.

A third case involves an attempted read access while the flash memory array is busy doing a write (program) or erase operation if the appropriate read-while-write control field is programmed for this response. The 3-bit read-while-write control allows for immediate termination of an attempted read, or various stall-while-write/erase operations are occurring.

### 30.8.6 Access pipelining

The platform flash memory controller does not support access pipelining since this capability is not supported by the flash memory array. As a result, the APC (Address Pipelining Control) field must be of the same value as the RWSC (Read Wait-State Control).

### 30.8.7 Flash error response operation

The flash memory array may signal an error response to terminate a requested access with an error. This may occur due to an uncorrectable ECC error, or because of improper sequencing during program/erase operations. When an error response is received, the platform flash memory controller does not update or validate a bank0 page read buffer nor the bank1 temporary holding register. An error response may be signaled on read or write operations. For additional information on the system registers which capture the faulting address, attributes, data and ECC information, see the chapter "Error Correction Status Module (ECSM)."

### 30.8.8 Bank0 page read buffers and prefetch operation

The logic associated with bank0 of the platform flash memory controller contains four 128-bit page read buffers which are used to hold instructions and data read from the flash memory array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

For the general case, a page buffer is written at the completion of an error-free flash memory access and the valid bit asserted. Subsequent flash memory accesses that "hit" the buffer, that is, the current access address matches the address stored in the buffer, can be serviced in 0 AHB wait-states as the stored read data is routed from the given page buffer back to the requesting bus master.

As noted in [Section 30.8.7: Flash error response operation](#), a page buffer is *not* marked as valid if the flash memory array access terminated with any type of transfer error. However, the result is that flash memory array accesses that are tagged with a single-bit correctable ECC event are loaded into the page buffer and validated. For additional comments on this topic, see [Section 30.8.8.4: Buffer invalidation](#).

Prefetch triggering is controllable on a per-master and access-type basis. Bus masters may be enabled or disabled from triggering prefetches, and triggering may be further restricted

based on whether a read access is for instruction or data. A read access to the platform flash memory controller may trigger a prefetch to the next sequential page of array data on the first idle cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a flash memory array prefetch is initiated if the data is not already resident in a page buffer. Prefetched data is always loaded into the least-recently-used buffer.

Buffers may be in one of six states, listed here in order of priority:

1. Invalid — The buffer contains no valid data.
2. Used — The buffer contains valid data which has been provided to satisfy an AHB burst type read.
3. Valid — The buffer contains valid data which has been provided to satisfy an AHB single type read.
4. Prefetched — The buffer contains valid data which has been prefetched to satisfy a potential future AHB access.
5. Busy AHB — The buffer is currently being used to satisfy an AHB burst read.
6. Busy Fill — The buffer has been allocated to receive data from the flash memory array, and the array access is still in progress.

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a simple numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least-recently-used buffer is selected for replacement.

Once the candidate page buffer has been selected, the flash memory array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, the recently-used status is not changed. Rather, it is marked as most-recently-used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of flash memory accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted flash memory access.

Several algorithms are available for prefetch control which trade off performance versus power. They are defined by the Bx\_Py\_PFLM (prefetch limit) register field. More aggressive prefetching increases power slightly due to the number of wasted (discarded) prefetches, but may increase performance by lowering average read latency.

In order for prefetching to occur, a number of control bits must be enabled. Specifically, the global buffer enable (PFCRn[Bx\_Py\_BFE]) must be set, the prefetch limit (PFCRn[Bx\_Py\_PFLM]) must be non-zero, either instruction prefetching (PFCRn[Bx\_Py\_IPFE]) or data prefetching (PFCRn[Bx\_Py\_DPF]) enabled, and Master Access must be enabled (PFAPR[MxPFD]). See [Section 30.7.2.2: Register description](#), for a description of these control fields.

### 30.8.8.1 Instruction/Data prefetch triggering

Prefetch triggering may be enabled for instruction reads via the Bx\_Py\_IPFE control field, while prefetching for data reads is enabled via the Bx\_Py\_DPF control field. Additionally,

the Bx\_Py\_PFLIM field must be set to enable prefetching. Prefetches are never triggered by write cycles.

### 30.8.8.2 Per-master prefetch triggering

Prefetch triggering may be also controlled for individual bus masters. See [Section 30.7.2.2.3: Platform Flash Access Protection Register \(PFAPR\)](#), for details on these controls.

### 30.8.8.3 Buffer allocation

Allocation of the line read buffers is controlled via page buffer configuration (Bx\_Py\_BCFG) field. This field defines the operating organization of the four page buffers. The buffers can be organized as a “pool” of available resources (with all four buffers in the pool) or with a fixed partition between buffers allocated to instruction or data accesses. For the fixed partition, two configurations are supported. In one configuration, buffers 0 and 1 are allocated for instruction fetches and buffers 2 and 3 for data accesses. In the second configuration, buffers 0, 1 and 2 are allocated for instruction fetches and buffer 3 reserved for data accesses.

### 30.8.8.4 Buffer invalidation

The page read buffers may be invalidated under hardware or software control.

At the beginning of all program/erase operations, the flash memory array will invalidate the page read buffers. Buffer invalidation occurs at the next AHB non-sequential access boundary, but does not affect a burst from a page read buffer which is in progress.

Software may invalidate the buffers by clearing the Bx\_Py\_BFE bit, which also disables the buffers. Software may then re-assert the Bx\_Py\_BFE bit to its previous state, and the buffers will have been invalidated.

One special case needing software invalidation relates to page buffer “hits” on flash memory data which was tagged with a single-bit ECC event on the original array access. Recall that the page buffer structure includes an status bit signaling the array access detected and corrected a single-bit ECC error. On all subsequent buffer hits to this type of page data, a single-bit ECC event is signaled by the platform flash memory controller. Depending on the specific hardware configuration, this reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

Finally, the buffers are invalidated by hardware on any non-sequential access with a non-zero value on haddr[28:24] to support wait-state emulation.

## 30.8.9 Bank1 Temporary Holding Register

Recall the bank1 logic within the platform flash memory controller includes a single 128-bit data register, used for capturing read data. Since this bank does not support prefetching, the read data for the referenced address is bypassed directly back to the AHB data bus. The page is also loaded into the temporary data register and subsequent accesses to this page can hit from this register, if it is enabled (B1\_P0\_BFE).

For the general case, a temporary holding register is written at the completion of an error-free flash memory access and the valid bit asserted. Subsequent flash memory accesses that “hit” the buffer, that is, the current access address matches the address stored in the



temporary holding register, can be serviced in 0 AHB wait-states as the stored read data is routed from the temporary register back to the requesting bus master.

The contents of the holding register are invalidated by the flash memory array at the beginning of all program/erase operations and on any non-sequential access with a non-zero value on `haddr[28:24]` (to support wait-state emulation) in the same manner as the bank0 page buffers. Additionally, the `B1_P0_BFE` register bit can be cleared by software to invalidate the contents of the holding register.

As noted in [Section 30.8.7: Flash error response operation](#), the temporary holding register is *not* marked as valid if the flash memory array access terminated with any type of transfer error. However, the result is that flash memory array accesses that are tagged with a single-bit correctable ECC event are loaded into the temporary holding register and validated. Accordingly, one special case needing software invalidation relates to holding register “hits” on flash memory data which was tagged with a single-bit ECC event. Depending on the specific hardware configuration, the reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

The bank1 temporary holding register effectively operates like a single page buffer.

### 30.8.10 Read-while-write functionality

The platform flash memory controller supports various programmable responses for read accesses while the flash memory is busy performing a write (program) or erase operation. For all situations, the platform flash memory controller uses the state of the flash memory array’s `MCR[DONE]` output to determine if it is busy performing some type of high voltage operation, namely, if `MCR[DONE] = 0`, the array is busy.

Specifically, two 3-bit read-while-write (`BKn_RWWC`) control register fields define the platform flash memory controller’s response to these types of access sequences. Five unique responses are defined by the `BKn_RWWC` setting: one that immediately reports an error on an attempted read and four settings that support various stall-while-write capabilities. Consider the details of these settings.

- `BKn_RWWC = 0b0--`  
For this mode, any attempted flash memory read to a busy array is immediately terminated with an AHB error response and the read is blocked in the controller and not seen by the flash memory array.
- `BKn_RWWC = 0b111`  
This defines the basic stall-while-write capability and represents the default reset setting. For this mode, the platform flash memory controller module simply stalls any read reference until the flash memory has completed its program/erase operation. If a read access arrives while the array is busy or if `MCR[DONE]` goes low while a read is still in progress, the AHB data phase is stalled and the read access address is saved. Once the array has completed its program/erase operation, the platform flash memory controller uses the saved address and attribute information to create a pseudo address phase cycle to “retry” the read reference and sends the registered information to the array. Once the retried address phase is complete, the read is processed normally and once the data is valid, it is forwarded to the AHB bus to terminate the system bus transfer.
- `BKn_RWWC = 0b110`  
This setting is similar to the basic stall-while-write capability provided when `BKn_RWWC = 0b111` with the added ability to generate a notification interrupt if a read

arrives while the array is busy with a program/erase operation. There are two notification interrupts, one for each bank (see the INTC chapter of this reference manual).

- BKn\_RWWC = 0b101

Again, this setting provides the basic stall-while-write capability with the added ability to abort any program/erase operation if a read access is initiated. For this setting, the read request is captured and retried as described for the basic stall-while-write, plus the program/erase operation is aborted by the platform flash memory controller. For this setting, no notification interrupts are generated.

- BKn\_RWWC = 0b100

This setting provides the basic stall-while-write capability with the ability to abort any program/erase operation if a read access is initiated plus the generation of an abort notification interrupt. For this setting, the read request is captured and retried as described for the basic stall-while-write, the program/erase operation is aborted by the platform flash memory controller and an abort notification interrupt generated. There are two abort notification interrupts, one for each bank.

As detailed above, a total of four interrupt requests are associated with the stall-while-write functionality. These interrupt requests are captured as part of ECSM's interrupt register and logically summed together to form a single request to the interrupt controller.

**Table 499. Platform flash memory controller stall-while-write interrupts**

MIR[n]	Interrupt description
ECSM.MIR[0]	Platform flash memory bank0 abort notification, MIR[FB0AI]
ECSM.MIR[1]	Platform flash memory bank0 stall notification, MIR[FB0SI]
ECSM.MIR[2]	Platform flash memory bank1 abort notification, MIR[FB1AI]
ECSM.MIR[3]	Platform flash memory bank1 stall notification, MIR[FB1S1]

### 30.8.11 Wait-state emulation

Emulation of other memory array timings are supported by the platform flash memory controller on read cycles to the flash memory. This functionality may be useful to maintain the access timing for blocks of memory which were used to overlay flash memory blocks for the purpose of system calibration or tuning during code development.

The platform flash memory controller inserts additional wait-states according to the values of haddr[28:24]. When these inputs are non-zero, additional cycles are added to AHB read cycles. Write cycles are not affected. In addition, no page read buffer prefetches are initiated, and buffer hits are ignored.

[Table 500](#) and [Table 501](#) show the relationship of haddr[28:24] to the number of additional primary wait-states. These wait-states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Note that the wait-state specification consists of two components: haddr[28:26] and haddr[25:24] and effectively extends the flash memory read by  $(8 * \text{haddr}[25:24] + \text{haddr}[28:26])$  cycles.

**Table 500. Additional wait-state encoding**

Memory address haddr[28:26]	Additional wait-states
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

*Table 501* shows the relationship of haddr[25:24] to the number of additional wait-states. These are applied in addition to those specified by haddr[28:26] and thus extend the total wait-state specification capability.

**Table 501. Extended additional wait-state encoding**

Memory address haddr[25:24]	Additional wait-states (added to those specified by haddr[28:26])
00	0
01	8
10	16
11	24

## 31 Static RAM (SRAM)

### 31.1 Introduction

This device has up to 96 KB of general-purpose static RAM (SRAM).

The SRAM provides the following features:

- SRAM can be read/written from any bus master
- Byte, halfword and word addressable
- ECC (error correction code) protected with single-bit correction and double-bit detection

**Table 502. SRAM behavior in chip modes**

SRAM — 96 KB	Address	MC_PCU control register	Run modes (DRUN, RUN0...3)	Low-power modes		
				HALT	STOP	STANDBY
8 KB	0x4000_0000 – 0x4000_1FFF	PCONF0[<mode>]	Always on	Always on	Always on	Always on
24 KB	0x4000_2000 – 0x4000_BFFF	PCONF2[<mode>]	On/off	On/off	On/off	On/off
64 KB	0x4000_C000 – 0x4001_7FFF	PCONF3[<mode>]	On/off	On/off	On/off	Always off

### 31.2 Low power configuration

In order to reduce leakage a portion of the SRAM can be switched off/unpowered during standby mode.

**Table 503. Low power configuration**

Mode	Configuration
RUN, TEST, SAFE and STOP	The entire SRAM is powered and operational.
STANDBY	Either 32 KB or just 8 KB of the SRAM remains powered. This option is software-selectable.

### 31.3 Register memory map

The L2SRAM occupies 96 KB of memory starting at the base address as shown in [Table 504](#).

**Table 504. SRAM memory map**

Address	Register name	Register description	Size
0x4000_0000 (Base)	—	SRA	96 KB

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM (see the *Error Correction Status Module (ECSM)* chapter of the reference manual for more information).

## 31.4 SRAM ECC mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 39-bit reads (32-bit data bus plus the 7-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

SRAM does not detect all errors greater than 2 bits.

Internal SRAM write operations are performed on the following byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32-bit data bus. The 8-bit ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 32-bit data width (1 or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1 or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data bus, and the 39-bit value is then written to SRAM.

### 31.4.1 Access timing

The system bus is a two-stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock cycle. [Table 505](#) lists the various combinations of read and write operations to SRAM and the number of wait states used for the each operation. The table columns contain the following information:

- Current operation — Lists the type of SRAM operation currently executing
- Previous operation — Lists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)
- Wait states — Lists the number of wait states (bus clocks) the operation requires which depends on the combination of the current and previous operation

**Table 505. Number of wait states required for SRAM operations**

Operation type	Current operation	Previous operation	Number of wait states required
Read	Read	Idle	1
		Pipelined read	
		8, 16 or 32-bit write	0 (read from the same address)
			1 (read from a different address)
	Pipelined read	Read	0
Write	8 or 16-bit write	Idle	1
		Read	
		Pipelined 8 or 16-bit write	2
		32-bit write	
		8 or 16-bit write	0 (write to the same address)
	Pipelined 8, 16 or 32-bit write	8, 16 or 32-bit write	0
	32-bit write	Idle	0
32-bit write			
Read			

### 31.4.2 Reset effects on SRAM accesses

Asynchronous reset will possibly corrupt SRAM if it asserts during a read or write operation to SRAM. The completion of that access depends on the cycle at which the reset occurs. Data read from or written to SRAM before the reset event occurred is retained, and no other address locations are accessed or changed. In case of no access ongoing when reset occurs, the SRAM corruption does not happen.

Instead, synchronous reset (SW reset) should be used in controlled function (without SRAM accesses) in case an initialization procedure without SRAM initialization is needed.

## 31.5 Functional description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a R/W operation. Because the ECC bits can contain random data after the device is powered on, the SRAM must be initialized by executing 32-bit write operations prior to any read accesses. This is also true for implicit read accesses caused by any write accesses of less than 32 bits as discussed in [Section 31.4, SRAM ECC mechanism](#).

## 31.6 Initialization and application information

To use the SRAM, the ECC must check all bits that require initialization after power on. All writes must specify an even number of registers performed on 32-bit word-aligned

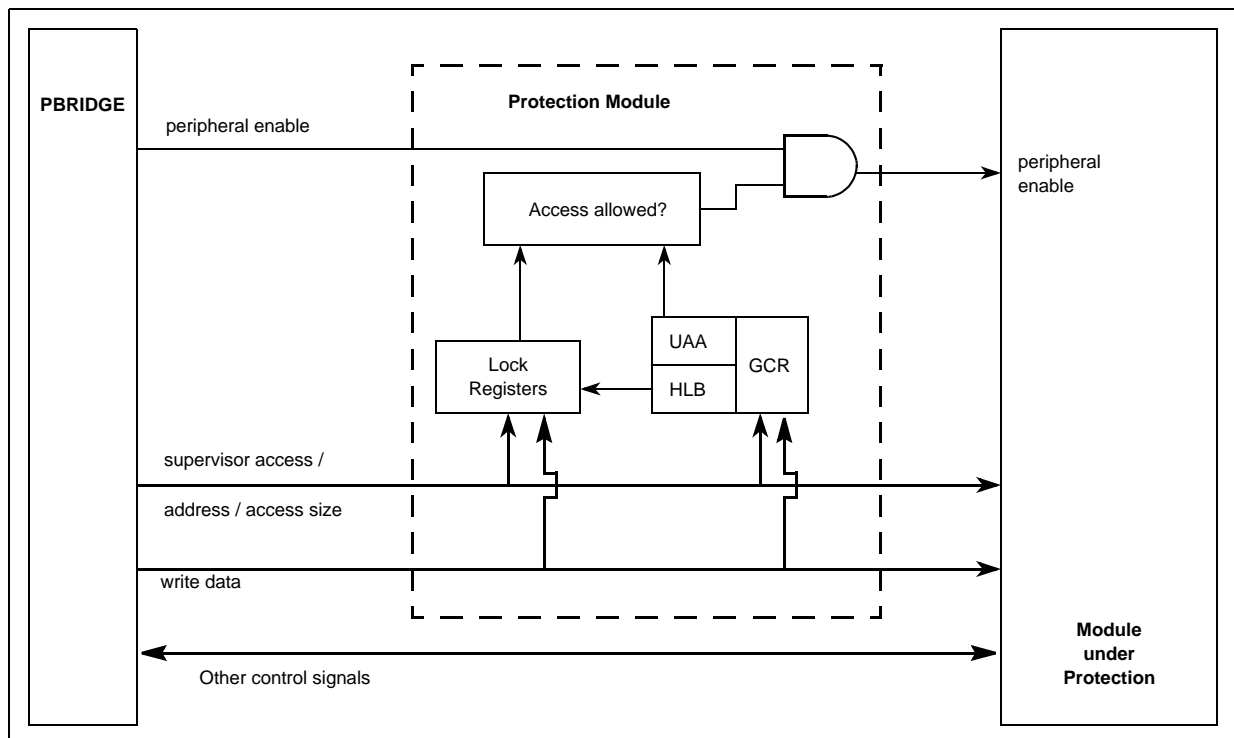
boundaries. If the write is not the entire 32 bits (8 or 16 bits), a read / modify / write operation is generated that checks the ECC value upon the read. See [Section 31.4, SRAM ECC mechanism](#).

## 32 Register Protection

### 32.1 Introduction

The Register Protection module offers a mechanism to protect defined memory-mapped address locations in a module under protection from being written. The address locations that can be protected are module-specific.

The protection module is located between the module under protection and the peripheral bridge. This is shown in *Figure 533*.



**Figure 533. Register Protection block diagram**

Please see the “Registers Under Protection” appendix for the list of protected registers.

### 32.2 Features

The Register Protection includes these distinctive features:

- Restrict write accesses for the module under protection to supervisor mode only
- Lock registers for first 6 KB of memory-mapped address space
- Address mirror automatically sets corresponding lock bit
- Once configured lock bits can be protected from changes



### 32.3 Modes of operation

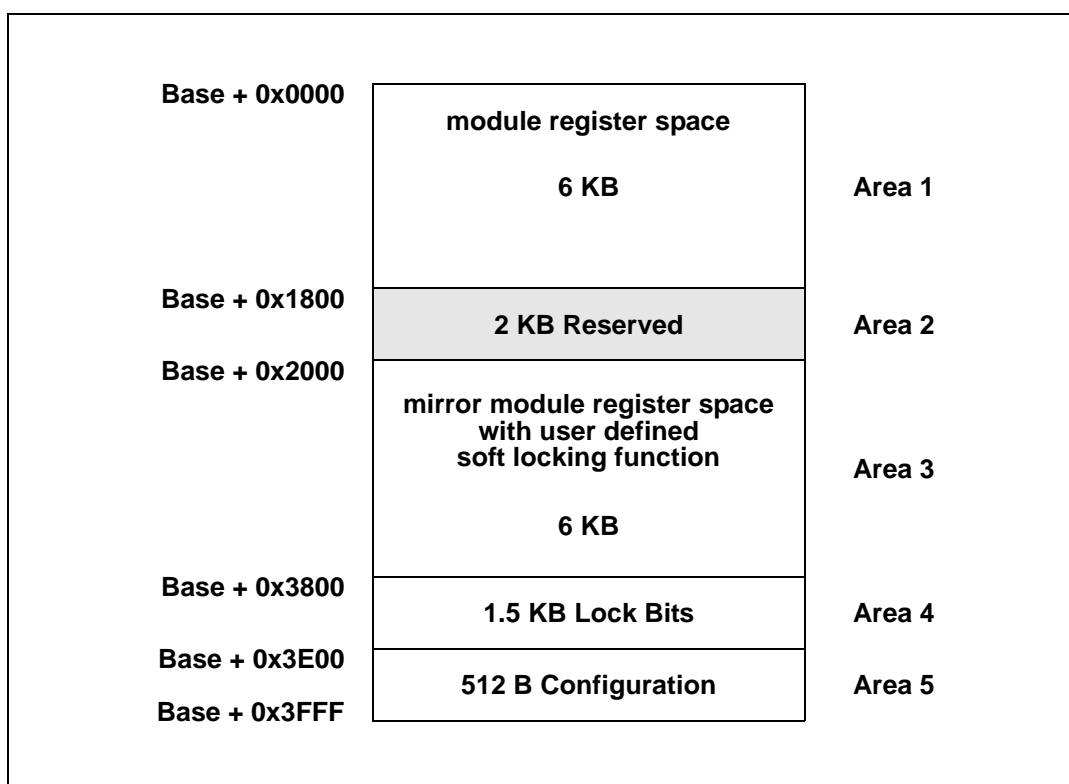
The Register Protection module is operable when the module under protection is operable.

### 32.4 External signal description

There are no external signals.

### 32.5 Memory map and register description

This section provides a detailed description of the memory map of a module using the Register Protection. The original 16 KB module memory space is divided into five areas as shown in [Figure 534](#).



**Figure 534. Register protection memory diagram**

Area 1 spans 6 KB and holds the normal functional module registers and is transparent for all read/write operations.

Area 2 spans 2 KB starting at address 0x1800. It is a reserved area, which cannot be accessed.

Area 3 spans 6 KB, starting at address 0x2000 and is a mirror of area 1. A read/write access to a 0x2000+X address will reads/writes the register at address X. As a side effect, a write access to address 0x2000+X sets the optional soft lock bits for address X in the same cycle as the register at address X is written. Not all registers in area 1 need to have protection defined by associated soft lock bits. For unprotected registers at address Y, accesses to

address 0x2000+Y will be identical to accesses at address Y. Only for registers implemented in area 1 and defined as protectable soft lock bits are available in area 4.

Area 4 is 1.5 KB and holds the soft lock bits, one bit per byte in area 1. The four soft lock bits associated with a module register word are arranged at byte boundaries in the memory map. The soft lock bit registers can be directly written using a bit mask.

Area 5 is 512 byte and holds the configuration bits of the protection mode. There is one configuration hard lock bit per module that prevents all further modifications to the soft lock bits and can only be cleared by a system reset once set. The other bits, if set, will allow user access to the protected module.

If any locked byte is accessed with a write transaction, a transfer error will be issued to the system and the write transaction will not be executed. This is true even if not all accessed bytes are locked.

Accessing unimplemented 32-bit registers in Areas 4 and 5 results in a transfer error.

### 32.5.1 Memory map

[Table 506](#) gives an overview on the Register Protection registers implemented.

**Table 506. Register protection memory map**

Address offset	Register	Location
0x0000	Module Register 0 (MR0)	<a href="#">on page 931</a>
0x0001	Module Register 1 (MR1)	<a href="#">on page 931</a>
0x0002	Module Register 2 (MR2)	<a href="#">on page 931</a>
0x0003–0x17FF	Module Register 3 (MR3) - Module Register 6143 (MR6143)	<a href="#">on page 931</a>
0x1800–0x1FFF	Reserved	—
0x2000	Module Register 0 (MR0) + Set soft lock bit 0 (LMR0)	<a href="#">on page 931</a>
0x2001	Module Register 1 (MR1) + Set soft lock bit 1 (LMR1)	<a href="#">on page 931</a>
0x2002–0x37FF	Module Register 2 (MR2) + Set soft lock bit 2 (LMR2) – Module Register 6143 (MR6143) + Set soft lock bit 6143 (LMR6143)	<a href="#">on page 931</a>
0x3800	Soft Lock Bit Register 0 (SLBR0): soft lock bits 0-3	<a href="#">on page 931</a>
0x3801	Soft Lock Bit Register 1 (SLBR1): soft lock bits 4-7	<a href="#">on page 931</a>
0x3802–0x3DFF	Soft Lock Bit Register 2 (SLBR2): soft lock bits 8-11 – Soft Lock Bit Register 1535 (SLBR1535): soft lock bits 6140-6143	<a href="#">on page 931</a>
0x3E00–0x3FFB	Reserved	—
0x3FFC	Global Configuration Register (GCR)	<a href="#">on page 932</a>

*Note:* Reserved registers in area #2 will be handled according to the protected IP (module under protection).

### 32.5.2 Register description

#### 32.5.2.1 Module Registers (MR0-6143)

This is the lower 6 KB module memory space which holds all the functional registers of the module that is protected by the Register Protection module.

#### 32.5.2.2 Module Register and Set Soft Lock Bit (LMR0-6143)

This is memory area #3 that provides mirrored access to the MR0-6143 registers with the side effect of setting soft lock bits in case of a write access to a MR that is defined as protectable by the locking mechanism. Each MR is protectable by one associated bit in a SLBR<sub>n</sub>.SLB<sub>m</sub>, according to the mapping described in [Table 507](#).

#### 32.5.2.3 Soft Lock Bit Register (SLBR0-1535)

These registers hold the soft lock bits for the protected registers in memory area #1.

Address 0x3800-0x3DFF Access: Read always  
Supervisor write

	0	1	2	3	4	5	6	7
R	0	0	0	0	SLB0	SLB1	SLB2	SLB3
W	WE0	WE1	WE2	WE3				
Reset	0	0	0	0	0	0	0	0

Figure 535. Soft Lock Bit Register (SLBR<sub>n</sub>)

Table 507. SLBR<sub>n</sub> field descriptions

Field	Description
WE0 WE1 WE2 WE3	Write Enable Bits for soft lock bits (SLB): WE0 enables writing to SLB0 WE1 enables writing to SLB1 WE2 enables writing to SLB2 WE3 enables writing to SLB3  1 Value is written to SLB 0 SLB is not modified
SLB0 SLB1 SLB2 SLB3	Soft lock bits for one MR <sub>n</sub> register: SLB0 can block accesses to MR[ <sub>n</sub> *4 + 0] SLB1 can block accesses to MR[ <sub>n</sub> *4 + 1] SLB2 can block accesses to MR[ <sub>n</sub> *4 + 2] SLB3 can block accesses to MR[ <sub>n</sub> *4 + 3]  1 Associated MR <sub>n</sub> byte is locked against write accesses 0 Associated MR <sub>n</sub> byte is unprotected and writeable

[Figure 508](#) gives some examples how SLBR<sub>n</sub>.SLB and MR<sub>n</sub> go together.

**Table 508. Soft lock bits vs. protected address**

Soft lock bit	Protected address
SLBR0.SLB0	MR0
SLBR0.SLB1	MR1
SLBR0.SLB2	MR2
SLBR0.SLB3	MR3
SLBR1.SLB0	MR4
SLBR1.SLB1	MR5
SLBR1.SLB2	MR6
SLBR1.SLB3	MR7
SLBR2.SLB0	MR8
...	...

**32.5.2.4 Global Configuration Register (GCR)**

This register is used to make global configurations related to register protection.

Address 0x3FFC

Access: Read Always Supervisor write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HLB	0	0	0	0	0	0	0	UAA	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 536. Global Configuration Register (GCR)**

**Table 509. GCR field descriptions**

Field	Description
HLB	Hard Lock Bit. This register can not be cleared once it is set by software. It can only be cleared by a system reset.  1 All SLB bits are write protected and can not be modified 0 All SLB bits are accessible and can be modified.
UAA	User Access Allowed.  1 The registers in the module under protection can be accessed in the mode defined for the module registers without any additional restrictions. 0 The registers in the module under protection can only be written in supervisor mode. All write accesses in non-supervisor mode are not executed and a transfer error is issued. This access restriction is in addition to any access restrictions imposed by the protected IP module.

*Note:* The GCR.UAA bit has no effect on the allowed access modes for the registers in the Register Protection module.

## 32.6 Functional description

### 32.6.1 General

This module provides a generic register (address) write-protection mechanism. The protection size can be:

- 32-bit (address == multiples of 4)
- 16-bit (address == multiples of 2)
- 8-bit (address == multiples of 1)
- unprotected (address == multiples of 1)

Which addresses are protected and the protection size depend on the SoC and/or module. Therefore this section can just give examples for various protection configurations.

For all addresses that are protected there are SLBR $n$ .SLB $m$  bits that specify whether the address is locked. When an address is locked it can be read but not written in any mode (supervisor/normal). If an address is unprotected the corresponding SLBR $n$ .SLB $m$  bit is always 0b0 no matter what software is writing to.

### 32.6.2 Change lock settings

To change the setting whether an address is locked or unlocked the corresponding SLBR $n$ .SLB $m$  bit needs to be changed. This can be done using the following methods:

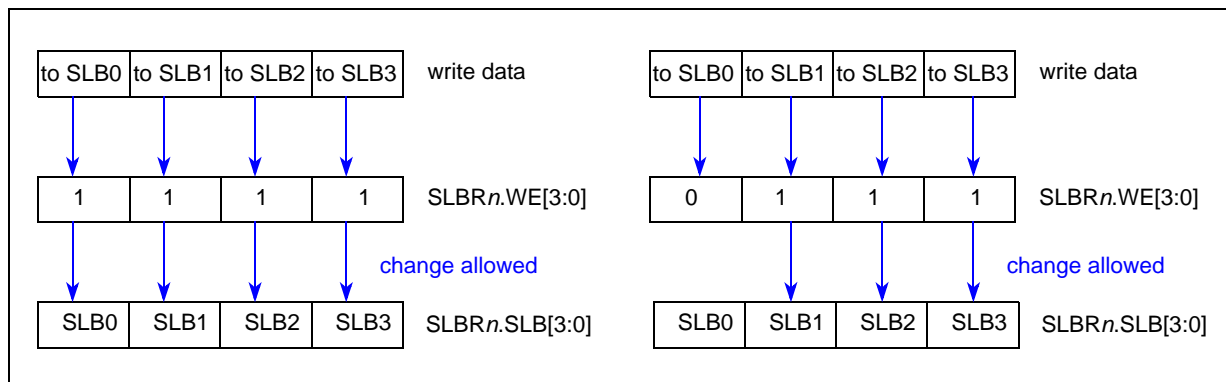
- Modify the SLBR $n$ .SLB $m$  directly by writing to area #4
- Set the SLBR $n$ .SLB $m$  bit(s) by writing to the mirror module space (area #3)

Both methods are explained in the following sections.

### 32.6.2.1 Change lock settings directly via area #4

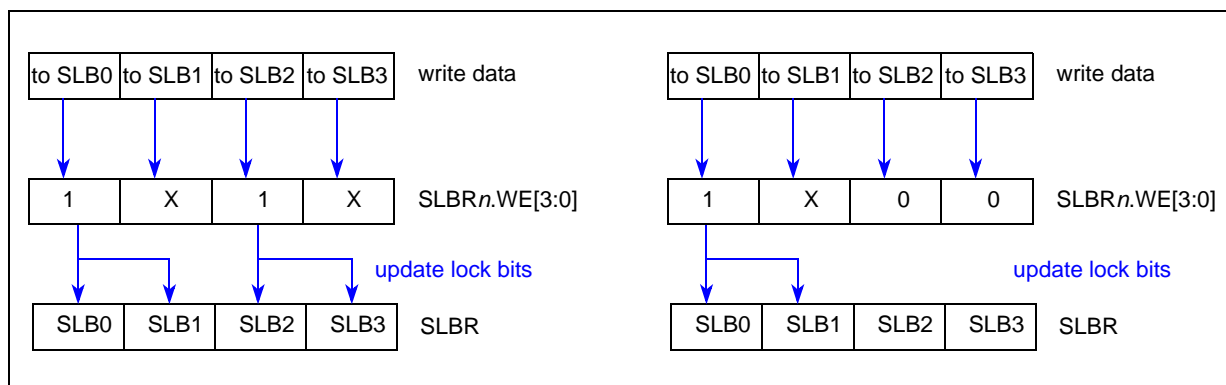
Memory area #4 contains the lock bits. They can be modified by writing to them. Each  $SLBRn.SLBm$  bit has a mask bit  $SLBRn.WEm$ , which protects it from being modified. This masking makes clear-modify-write operations unnecessary.

*Figure 537* shows two modification examples. In the left example there is a write access to the  $SLBRn$  register specifying a mask value which allows modification of all  $SLBRn.SLBm$  bits. The example on the right specifies a mask which only allows modification of the bits  $SLBRn.SLB[3:1]$ .



**Figure 537. Change Lock Settings Directly Via Area #4**

*Figure 537* shows four registers that can be protected 8-bit wise. In *Figure 538* registers with 16-bit protection and in *Figure 539* registers with 32-bit protection are shown:

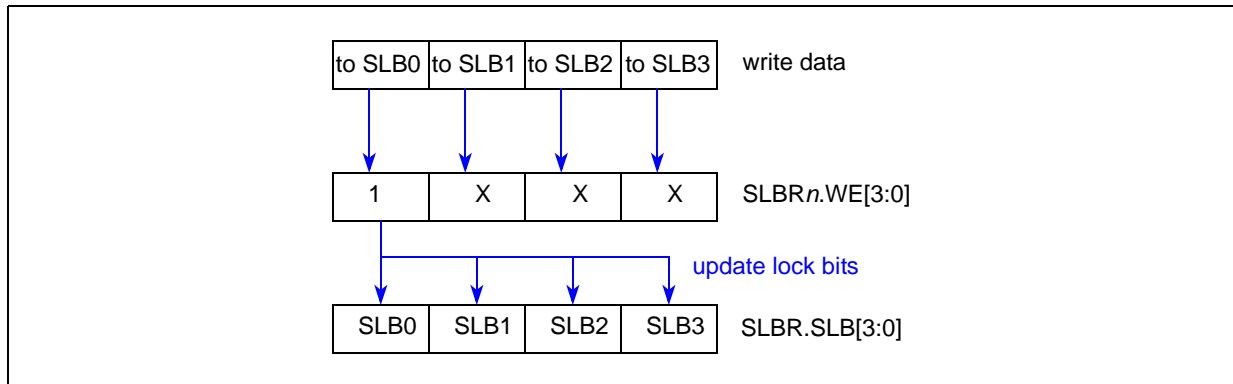


**Figure 538. Change Lock Settings for 16-bit Protected Addresses**

On the right side of *Figure 538* it is shown that the data written to  $SLBRn.SLB[0]$  is automatically written to  $SLBRn.SLB[1]$  also. This is done as the address reflected by  $SLBRn.SLB[0]$  is protected 16-bit wise. Note that in this case the write enable  $SLBRn.WE[0]$  must be set while  $SLBRn.WE[1]$  does not matter. As the enable bits  $SLBRn.WE[3:2]$  are cleared the lock bits  $SLBRn.SLB[3:2]$  remain unchanged.

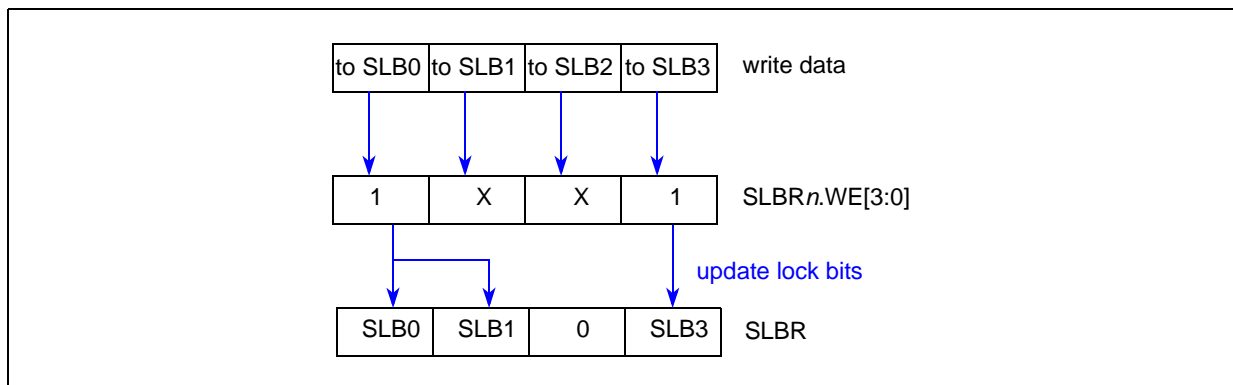
In the example on the left side of *Figure 538* the data written to  $SLBRn.SLB[0]$  is mirrored to  $SLBRn.SLB[1]$  and the data written to  $SLBRn.SLB[2]$  is mirrored to  $SLBRn.SLB[3]$  as for both registers the write enables are set.

In [Figure 539](#) a 32-bit wise protected register is shown. When SLBRn.WE[0] is set the data written to SLBRn.SLB[0] is automatically written to SLBRn.SLB[3:1] also. Otherwise SLBRn.SLB[3:0] remains unchanged.



**Figure 539. Change Lock Settings for 32-bit Protected Addresses**

In [Figure 540](#) an example is shown which has a mixed protection size configuration:

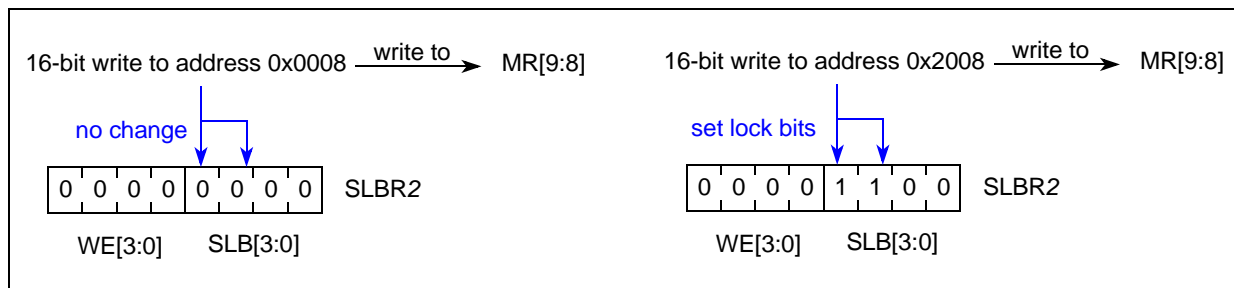


**Figure 540. Change Lock Settings for Mixed Protection**

The data written to SLBRn.SLB[0] is mirrored to SLBRn.SLB[1] as the corresponding register is 16-bit protected. The data written to SLBRn.SLB[2] is blocked as the corresponding register is unprotected. The data written to SLBRn.SLB[3] is written to SLBRn.SLB[3].

### 32.6.2.2 Enable locking via mirror module space (area #3)

It is possible to enable locking for a register after writing to it. To do so the mirrored module address space must be used. [Figure 541](#) shows one example:

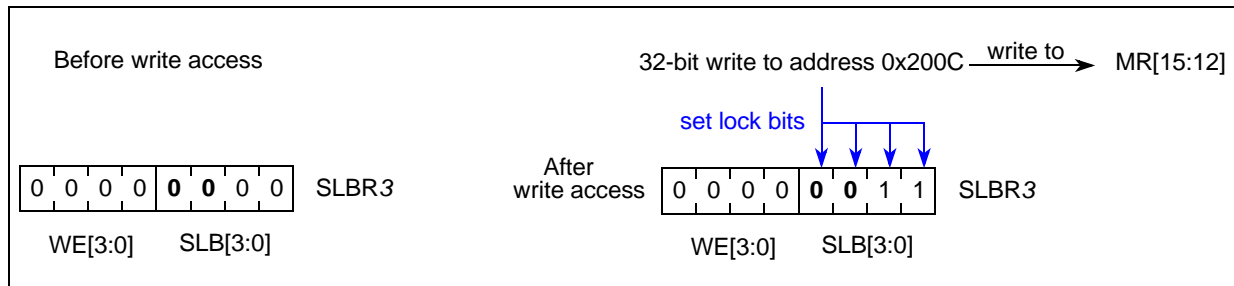


**Figure 541. Enable Locking Via Mirror Module Space (Area #3)**

When writing to address 0x0008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits remain unchanged (left part of [Figure 538](#)).

When writing to address 0x2008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits SLBR2.SLB[1:0] are set while the lock bits SLBR2.SLB[3:2] remain unchanged (right part of [Figure 538](#)).

[Figure 542](#) shows an example where some addresses are protected and some are not:



**Figure 542. Enable Locking for Protected and Unprotected Addresses**

In the example in [Figure 542](#) addresses 0x0C and 0x0D are unprotected. Therefore their corresponding lock bits SLBR3.SLB[1:0] are always 0b0 (shown in bold). When doing a 32-bit write access to address 0x200C only lock bits SLBR3.SLB[3:2] are set while bits SLBR3.SLB[1:0] stay 0b0.

*Note:* Lock bits can only be set via writes to the mirror module space. Reads from the mirror module space will not change the lock bits.

**32.6.2.3 Write protection for locking bits**

Changing the locking bits through any of the procedures mentioned in [Section 32.6.2.1: Change lock settings directly via area #4](#) and [Section 32.6.2.2: Enable locking via mirror module space \(area #3\)](#) is only possible as long as the bit GCR.HLB is cleared. Once this bit is set the locking bits can no longer be modified until there is a system reset.

**32.6.3 Access errors**

The protection module generates transfer errors under several circumstances. For the area definition refer to [Figure 534](#).



1. If accessing area #1 or area #3, the protection module transfers any access error from the underlying Module under Protection.
2. If user mode is not allowed, user write attempts to all areas will assert a transfer error and the writes will be blocked.
3. Access attempts to the reserved area #2 cause a transfer error to be asserted.
4. Access attempts to unimplemented 32-bit registers in area #4 or area #5 cause a transfer error to be asserted.
5. Attempted writes to a register in area #1 or area #3 with soft lock bit set for any of the affected bytes causes a transfer error to be asserted and the write is blocked. The complete write operation to non-protected bytes in this word is ignored.
6. If writing to a soft lock register in area #4 with the hard lock bit being set a transfer error is asserted.
7. Any write operation in any access mode to area #3 while GCR.HLB is set result in a error.

### 32.7 Reset

The reset state of each individual bit is shown within the Register Description section (See [Section 32.5.2: Register description](#)). In summary, after reset, locking for all MRn registers is disabled. The registers can be accessed in Supervisor Mode only.

### 32.8 Protected registers

For SPC560B54/6x the Register Protection module protects the registers shown in [Table 510](#).

**Table 510. Protected registers**

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
<b>Code flash memory 0, 4 registers to protect</b>					
Code Flash 0	MCR	32	C3F88000	000	bits[0:31]
Code Flash 0	PFCR0	32	C3F88000	01C	bits[0:31]
Code Flash 0	PFCR1	32	C3F88000	020	bits[0:31]
Code Flash 0	PFAPR	32	C3F88000	024	bits[0:31]
<b>Data flash memory, 1 register to protect</b>					
Data Flash	MCR	32	C3F8C000	000	bits[0:31]
<b>Code Flash 1, 2 registers to protect</b>					
Code Flash 1	MCR	32	C3FB0000	000	bits[0:31]
Code Flash 2	MCR	32	C3FB4000	000	bits[0:31]
<b>SIU lite, 64 registers to protect</b>					
SIUL	IRER	32	C3F90000	018	bits[0:31]

**Table 510. Protected registers (continued)**

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
SIUL	IREER	32	C3F90000	028	bits[0:31]
SIUL	IFEER	32	C3F90000	02C	bits[0:31]
SIUL	IFER	32	C3F90000	030	bits[0:31]
SIUL	PCR0	16	C3F90000	040	bits[0:15]
SIUL	PCR1	16	C3F90000	042	bits[0:15]
SIUL	PCR2	16	C3F90000	044	bits[0:15]
SIUL	PCR3	16	C3F90000	046	bits[0:15]
SIUL	PCR4	16	C3F90000	048	bits[0:15]
SIUL	PCR5	16	C3F90000	04A	bits[0:15]
SIUL	PCR6	16	C3F90000	04C	bits[0:15]
SIUL	PCR7	16	C3F90000	04E	bits[0:15]
SIUL	PCR8	16	C3F90000	050	bits[0:15]
SIUL	PCR9	16	C3F90000	052	bits[0:15]
SIUL	PCR10	16	C3F90000	054	bits[0:15]
SIUL	PCR11	16	C3F90000	056	bits[0:15]
SIUL	PCR12	16	C3F90000	058	bits[0:15]
SIUL	PCR13	16	C3F90000	05A	bits[0:15]
SIUL	PCR14	16	C3F90000	05C	bits[0:15]
SIUL	PCR15	16	C3F90000	05E	bits[0:15]
SIUL	PCR16	16	C3F90000	060	bits[0:15]
SIUL	PCR17	16	C3F90000	062	bits[0:15]
SIUL	PCR18	16	C3F90000	064	bits[0:15]
SIUL	PCR19	16	C3F90000	066	bits[0:15]
SIUL	PCR34	16	C3F90000	084	bits[0:15]
SIUL	PCR35	16	C3F90000	086	bits[0:15]
SIUL	PCR36	16	C3F90000	088	bits[0:15]
SIUL	PCR37	16	C3F90000	08A	bits[0:15]
SIUL	PCR38	16	C3F90000	08C	bits[0:15]
SIUL	PCR39	16	C3F90000	08E	bits[0:15]
SIUL	PCR40	16	C3F90000	090	bits[0:15]
SIUL	PCR41	16	C3F90000	092	bits[0:15]
SIUL	PCR42	16	C3F90000	094	bits[0:15]
SIUL	PCR43	16	C3F90000	096	bits[0:15]
SIUL	PCR44	16	C3F90000	098	bits[0:15]



Table 510. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
SIUL	PCR45	16	C3F90000	09A	bits[0:15]
SIUL	PCR46	16	C3F90000	09C	bits[0:15]
SIUL	PCR47	16	C3F90000	09E	bits[0:15]
SIUL	PCR64	16	C3F90000	0C0	bits[0:15]
SIUL	PCR65	16	C3F90000	0C2	bits[0:15]
SIUL	PCR66	16	C3F90000	0C4	bits[0:15]
SIUL	PCR67	16	C3F90000	0C6	bits[0:15]
SIUL	PCR68	16	C3F90000	0C8	bits[0:15]
SIUL	PCR69	16	C3F90000	0CA	bits[0:15]
SIUL	PCR70	16	C3F90000	0CC	bits[0:15]
SIUL	PCR71	16	C3F90000	0CE	bits[0:15]
SIUL	PCR72	16	C3F90000	0D0	bits[0:15]
SIUL	PCR73	16	C3F90000	0D2	bits[0:15]
SIUL	PCR74	16	C3F90000	0D4	bits[0:15]
SIUL	PCR75	16	C3F90000	0D6	bits[0:15]
SIUL	PCR76	16	C3F90000	0D8	bits[0:15]
SIUL	PCR77	16	C3F90000	0DA	bits[0:15]
SIUL	PCR78	16	C3F90000	0DC	bits[0:15]
SIUL	PCR79	16	C3F90000	0DE	bits[0:15]
SIUL	PCR88	16	C3F90000	0F0	bits[0:15]
SIUL	PCR89	16	C3F90000	0F2	bits[0:15]
SIUL	PCR90	16	C3F90000	0F4	bits[0:15]
SIUL	PCR91	16	C3F90000	0F6	bits[0:15]
SIUL	PCR92	16	C3F90000	0F8	bits[0:15]
SIUL	PCR93	16	C3F90000	0FA	bits[0:15]
SIUL	PCR94	16	C3F90000	0FC	bits[0:15]
SIUL	PCR95	16	C3F90000	0FE	bits[0:15]
SIUL	PCR96	16	C3F90000	100	bits[0:15]
SIUL	PCR97	16	C3F90000	102	bits[0:15]
SIUL	PCR98	16	C3F90000	104	bits[0:15]
SIUL	PCR99	16	C3F90000	106	bits[0:15]
SIUL	PCR100	16	C3F90000	108	bits[0:15]
SIUL	PCR101	16	C3F90000	10A	bits[0:15]
SIUL	PCR102	16	C3F90000	10C	bits[0:15]

Table 510. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
SIUL	PCR103	16	C3F90000	10E	bits[0:15]
SIUL	PCR104	16	C3F90000	110	bits[0:15]
SIUL	PCR105	16	C3F90000	112	bits[0:15]
SIUL	PCR106	16	C3F90000	114	bits[0:15]
SIUL	PCR107	16	C3F90000	116	bits[0:15]
SIUL	PCR108	16	C3F90000	118	bits[0:15]
SIUL	PCR109	16	C3F90000	11A	bits[0:15]
SIUL	PCR110	16	C3F90000	11C	bits[0:15]
SIUL	PCR111	16	C3F90000	11E	bits[0:15]
SIUL	PCR112	16	C3F90000	120	bits[0:15]
SIUL	PCR113	16	C3F90000	122	bits[0:15]
SIUL	PCR114	16	C3F90000	124	bits[0:15]
SIUL	PCR115	16	C3F90000	126	bits[0:15]
SIUL	PCR123	16	C3F90000	136	bits[0:15]
SIUL	PCR124	16	C3F90000	138	bits[0:15]
SIUL	PCR125	16	C3F90000	13A	bits[0:15]
SIUL	PCR126	16	C3F90000	13C	bits[0:15]
SIUL	PCR127	16	C3F90000	13E	bits[0:15]
SIUL	PCR128	16	C3F90000	140	bits[0:15]
SIUL	PCR129	16	C3F90000	142	bits[0:15]
SIUL	PCR130	16	C3F90000	144	bits[0:15]
SIUL	PCR131	16	C3F90000	146	bits[0:15]
SIUL	PCR132	16	C3F90000	148	bits[0:15]
SIUL	PCR133	16	C3F90000	14A	bits[0:15]
SIUL	PCR134	16	C3F90000	14C	bits[0:15]
SIUL	PCR135	16	C3F90000	14E	bits[0:15]
SIUL	PSMI0_3	8	C3F90000	500	bits[0:7]
SIUL	PSMI4_7	8	C3F90000	504	bits[0:7]
SIUL	PSMI8_11	8	C3F90000	508	bits[0:7]
SIUL	PSMI12_15	8	C3F90000	50C	bits[0:7]
SIUL	PSMI16_19	8	C3F90000	510	bits[0:7]
SIUL	PSMI20_23	32	C3F90000	514	bits[0:7]
SIUL	PSMI24_27	32	C3F90000	518	bits[0:7]
SIUL	PSMI28_31	32	C3F90000	51C	bits[0:7]

**Table 510. Protected registers (continued)**

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
SIUL	PSMI32_35	32	C3F90000	520	bits[0:7]
SIUL	PSMI36_39	32	C3F90000	524	bits[0:7]
SIUL	PSMI40_43	32	C3F90000	528	bits[0:7]
SIUL	PSMI44_47	32	C3F90000	52C	bits[0:7]
SIUL	PSMI48_51	32	C3F90000	530	bits[0:7]
SIUL	PSMI52_55	32	C3F90000	534	bits[0:7]
SIUL	PSMI56_59	32	C3F90000	538	bits[0:7]
SIUL	PSMI61_63	32	C3F90000	53C	bits[0:7]
SIUL	IFMC0	32	C3F90000	1000	bits[0:31]
SIUL	IFMC1	32	C3F90000	1004	bits[0:31]
SIUL	IFMC2	32	C3F90000	1008	bits[0:31]
SIUL	IFMC3	32	C3F90000	100C	bits[0:31]
SIUL	IFMC4	32	C3F90000	1010	bits[0:31]
SIUL	IFMC5	32	C3F90000	1014	bits[0:31]
SIUL	IFMC6	32	C3F90000	1018	bits[0:31]
SIUL	IFMC7	32	C3F90000	101C	bits[0:31]
SIUL	IFMC8	32	C3F90000	1020	bits[0:31]
SIUL	IFMC9	32	C3F90000	1024	bits[0:31]
SIUL	IFMC10	32	C3F90000	1028	bits[0:31]
SIUL	IFMC11	32	C3F90000	102C	bits[0:31]
SIUL	IFMC12	32	C3F90000	1030	bits[0:31]
SIUL	IFMC13	32	C3F90000	1034	bits[0:31]
SIUL	IFMC14	32	C3F90000	1038	bits[0:31]
SIUL	IFMC15	32	C3F90000	103C	bits[0:31]
SIUL	IFMC16	32	C3F90000	1040	bits[0:31]
SIUL	IFMC17	32	C3F90000	1044	bits[0:31]
SIUL	IFMC18	32	C3F90000	1048	bits[0:31]
SIUL	IFMC19	32	C3F90000	104C	bits[0:31]
SIUL	IFMC20	32	C3F90000	1050	bits[0:31]
SIUL	IFMC21	32	C3F90000	1054	bits[0:31]
SIUL	IFMC22	32	C3F90000	1058	bits[0:31]
SIUL	IFMC23	32	C3F90000	105C	bits[0:31]
SIUL	IFCPR	32	C3F90000	1080	bits[0:31]
<b>Mode Entry Module, 41 registers to protect</b>					

Table 510. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
MC ME	ME_ME	32	C3FDC000	008	bits[0:31]
MC ME	ME_IM	32	C3FDC000	010	bits[0:31]
MC ME	ME_TEST_MC	32	C3FDC000	024	bits[0:31]
MC ME	ME_SAFE_MC	32	C3FDC000	028	bits[0:31]
MC ME	ME_DRUN_MC	32	C3FDC000	02C	bits[0:31]
MC ME	ME_RUN0_MC	32	C3FDC000	030	bits[0:31]
MC ME	ME_RUN1_MC	32	C3FDC000	034	bits[0:31]
MC ME	ME_RUN2_MC	32	C3FDC000	038	bits[0:31]
MC ME	ME_RUN3_MC	32	C3FDC000	03C	bits[0:31]
MC ME	ME_HALT_MC	32	C3FDC000	040	bits[0:31]
MC ME	ME_STOP_MC	32	C3FDC000	048	bits[0:31]
MC ME	ME_STANDBY_MC	32	C3FDC000	054	bits[0:31]
MC ME	ME_RUN_PC0	32	C3FDC000	080	bits[0:31]
MC ME	ME_RUN_PC1	32	C3FDC000	084	bits[0:31]
MC ME	ME_RUN_PC2	32	C3FDC000	088	bits[0:31]
MC ME	ME_RUN_PC3	32	C3FDC000	08C	bits[0:31]
MC ME	ME_RUN_PC4	32	C3FDC000	090	bits[0:31]
MC ME	ME_RUN_PC5	32	C3FDC000	094	bits[0:31]
MC ME	ME_RUN_PC6	32	C3FDC000	098	bits[0:31]
MC ME	ME_RUN_PC7	32	C3FDC000	09C	bits[0:31]
MC ME	ME_LP_PC0	32	C3FDC000	0A0	bits[0:31]
MC ME	ME_LP_PC1	32	C3FDC000	0A4	bits[0:31]
MC ME	ME_LP_PC2	32	C3FDC000	0A8	bits[0:31]
MC ME	ME_LP_PC3	32	C3FDC000	0AC	bits[0:31]
MC ME	ME_LP_PC4	32	C3FDC000	0B0	bits[0:31]
MC ME	ME_LP_PC5	32	C3FDC000	0B4	bits[0:31]
MC ME	ME_LP_PC6	32	C3FDC000	0B8	bits[0:31]
MC ME	ME_LP_PC7	32	C3FDC000	0BC	bits[0:31]
MC ME	ME_PCTL[4..7]	32	C3FDC000	0C4	bits[0:31]
MC ME	ME_PCTL[16..19]	32	C3FDC000	0D0	bits[0:31]
MC ME	ME_PCTL[20..23]	32	C3FDC000	0D4	bits[0:31]
MC ME	ME_PCTL[32..35]	32	C3FDC000	0E0	bits[0:31]
MC ME	ME_PCTL[44..47]	32	C3FDC000	0EC	bits[0:31]
MC ME	ME_PCTL[48..51]	32	C3FDC000	0F0	bits[0:31]

Table 510. Protected registers (continued)

Module	Register	Protected size (bits)	Module base address	Register offset	Protected bits
MC ME	ME_PCTL[56..59]	32	C3FDC000	0F8	bits[0:31]
MC ME	ME_PCTL[60..63]	32	C3FDC000	0FC	bits[0:31]
MC ME	ME_PCTL[68..71]	32	C3FDC000	104	bits[0:31]
MC ME	ME_PCTL[72..75]	32	C3FDC000	108	bits[0:31]
MC ME	ME_PCTL[88..91]	32	C3FDC000	118	bits[0:31]
MC ME	ME_PCTL[92..95]	32	C3FDC000	11C	bits[0:31]
MC ME	ME_PCTL[104..107]	32	C3FDC000	128	bits[0:31]
<b>Clock Generation Module, 3 registers to protect</b>					
MC CGM	CGM_OC_EN	8	C3FE0000	373	bits[0:7]
MC CGM	CGM_OCDS_SC	8	C3FE0000	374	bits[0:7]
MC CGM	CGM_SC_DC[0..3]	32	C3FE0000	37C	bits[0:31]
<b>CMU, 1 register to protect</b>					
CMU	CMU_CSR	8	C3FE0000	103	bits[24:31]
<b>Reset Generation Module, 7 registers to protect</b>					
MC RGM	RGM_FERD	16	C3FE4000	004	bits[0:15]
MC RGM	RGM_DERD	16	C3FE4000	006	bits[0:15]
MC RGM	RGM_FEAR	16	C3FE4000	010	bits[0:15]
MC RGM	RGM_DEAR	16	C3FE4000	012	bits[0:15]
MC RGM	RGM_FESS	16	C3FE4000	018	bits[0:15]
MC RGM	RGM_STDBY	16	C3FE4000	01A	bits[0:15]
MC RGM	RGM_FBRE	16	C3FE4000	01C	bits[0:15]
<b>Power Control Unit, 2 registers to protect</b>					
MC PCU	PCONF2	32	C3FE8000	008	bits[0:31]
MC PCU	PCONF3	32	C3FE8000	00C	bits[0:31]

## 33 Software Watchdog Timer (SWT)

### 33.1 Overview

The SWT is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing sequence. Writing the sequence resets the timer to a specified time-out period. If this servicing action does not occur before the timer expires the SWT generates an interrupt or hardware reset. The SWT can be configured to generate a reset or interrupt on an initial time-out, a reset is always generated on a second consecutive time-out.

The SWT provides a window functionality. When this functionality is programmed, the servicing action should take place within the defined window. When occurring outside the defined period, the SWT generates a reset.

### 33.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- The unique SWT counter clock is the undivided slow internal RC oscillator 128 kHz (SIRC), no other clock source can be selected
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Master access protection
- Hard and soft configuration lock bits
- The SWT is started on exit of power-on phase (RGM phase 2) to monitor flash boot sequence phase. It is then reset during RGM phase3 and optionally enabled when platform reset is released depending on value of flash user option bit 31 (WATCHDOG\_EN).

### 33.3 Modes of operation

The SWT supports three device modes of operation: normal, debug and stop. When the SWT is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the SWT\_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run. In STOP mode, operation of the counter is controlled by the STP bit in the SWT\_CR. If the STP bit is set, the counter is stopped in STOP mode, otherwise it continues to run. On exit from STOP mode, the SWT will continue from the state it was before entering this mode.

The software watchdog is not available during standby. On exit from standby, the SWT behaves in a usual “out of reset” situation.

### 33.4 External signal description

The SWT module does not have any external interface signals.



## 33.5 Memory map and register description

The SWT programming model has six 32-bit registers. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include: writes to read only registers, incorrect values written to the service register when enabled, accesses to reserved addresses and accesses by masters without permission. If the SWT\_CR[RIA] bit is set, then the SWT generates a system reset on an invalid access otherwise a bus error is generated. If either the HLK or SLK bits in the SWT\_CR are set then the SWT\_CR, SWT\_TO and SWT\_WN registers are read only.

### 33.5.1 Memory map

The SWT memory map is shown in [Table 511](#). The reset values of SWT\_CR, SWT\_TO and SWT\_WN are device specific. These values are determined by SWT inputs.

**Table 511. SWT memory map**

Base address: 0xFFF3_8000		
Address offset	Register	Location
0x0000	SWT Control Register (SWT_CR)	<a href="#">on page 945</a>
0x0004	SWT Interrupt Register (SWT_IR)	<a href="#">on page 947</a>
0x0008	SWT Time-Out Register (SWT_TO)	<a href="#">on page 948</a>
0x000C	SWT Window Register (SWT_WN)	<a href="#">on page 948</a>
0x0010	SWT Service Register (SWT_SR)	<a href="#">on page 949</a>
0x0014	SWT Counter Output Register (SWT_CO)	<a href="#">on page 949</a>

### 33.5.2 Register description

#### 33.5.2.1 SWT Control Register (SWT\_CR)

The SWT\_CR contains fields for configuring and controlling the SWT. The reset value of this register is device specific. Some devices can be configured to automatically clear the SWT\_CR.WEN bit during the boot process. This register is read only if either the SWT\_CR.HLK or SWT\_CR.SLK bits are set.

Offset 0x0000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP	MAP	MAP	MAP	MAP	MAP	MAP	MAP	0	0	0	0	0	0	0	0
W	0	1	2	3	4	5	6	7								
Reset <sup>1</sup>	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	CSL	STP	FRZ	WEN
W																
Reset <sup>1</sup>	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	1

1. The reset value for the SWT\_CR is device specific.

**Figure 543. SWT Control Register (SWT\_CR)**

Default value for SWT\_CR\_RST is 0x4000\_011B, corresponding to MAP1 = 1 (only data bus access allowed), RIA = 1 (reset on invalid SWT access), SLK = 1 (soft lock), CSL = 1 (IRC clock source for counter), FRZ = 1 (freeze on debug), WEN = 1 (watchdog enable). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG\_EN) is '0'.

**Table 512. SWT\_CR field descriptions**

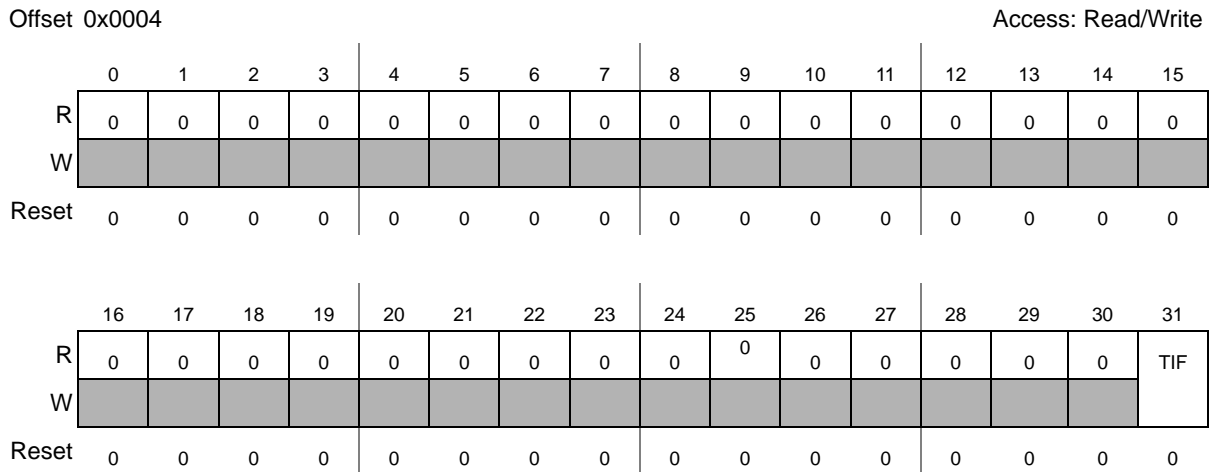
Field	Description
MAPn	Master Access Protection for Master n. The platform bus master assignments are device specific. 0 = Access for the master is not enabled 1 = Access for the master is enabled
KEY	Keyed Service Mode. 0 = Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 is used to service the watchdog 1 = Keyed Service Mode, two pseudorandom key value are used to service the watchdog
RIA	Reset on Invalid Access. 0 = Invalid access to the SWT generates a bus error 1 = Invalid access to the SWT causes a system reset if WEN=1
WND	Window Mode. 0 = Regular mode, service sequence can be done at any time 1 = Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.
ITR	Interrupt Then Reset. 0 = Generate a reset on a time-out 1 = Generate an interrupt on an initial time-out, reset on a second consecutive time-out
HLK	Hard Lock. This bit is only cleared at reset. 0 = SWT_CR, SWT_TO and SWT_WN are read/write registers if SLK=0 1 = SWT_CR, SWT_TO and SWT_WN are read only registers

**Table 512. SWT\_CR field descriptions**

Field	Description
SLK	Soft Lock. This bit is cleared by writing the unlock sequence to the service register. 0 = SWT_CR, SWT_TO and SWT_WN are read/write registers if HLK=0 1 = SWT_CR, SWT_TO and SWT_WN are read only registers
CSL	Clock Selection. Selects the SIRC oscillator clock that drives the internal timer. CSL bit can be written. The status of the bit has no effect on counter clock selection on SPC560B54/6x device. 0 = System clock (Not applicable in SPC560B54/6x) 1 = Oscillator clock
STP	Stop Mode Control. Allows the watchdog timer to be stopped when the device enters STOP mode. 0 = SWT counter continues to run in STOP mode 1 = SWT counter is stopped in STOP mode
FRZ	Debug Mode Control. Allows the watchdog timer to be stopped when the device enters debug mode. 0 = SWT counter continues to run in debug mode 1 = SWT counter is stopped in debug mode
WEN	Watchdog Enabled. 0 = SWT is disabled 1 = SWT is enabled

**33.5.2.2 SWT Interrupt Register (SWT\_IR)**

The SWT\_IR contains the time-out interrupt flag.



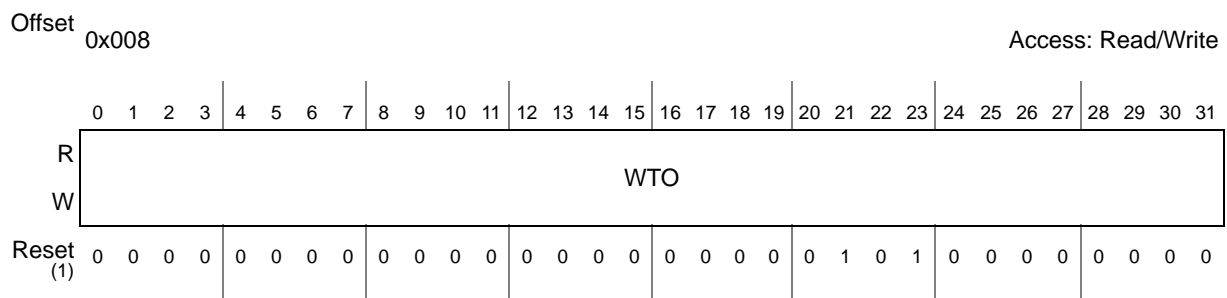
**Figure 544. SWT Interrupt Register (SWT\_IR)**

**Table 513. SWT\_IR field descriptions**

Field	Description
TIF	Time-out Interrupt Flag. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 = No interrupt request 1 = Interrupt request due to an initial time-out

**33.5.2.3 SWT Time-Out Register (SWT\_TO)**

The SWT Time-Out (SWT\_TO) register contains the 32-bit time-out period. The reset value for this register is device specific. This register is read only if either the SWT\_CR.HLK or SWT\_CR.SLK bits are set.



**Figure 545. SWT Time-Out Register (SWT\_TO)**

1. The reset value of the SWT\_TO register is device specific.

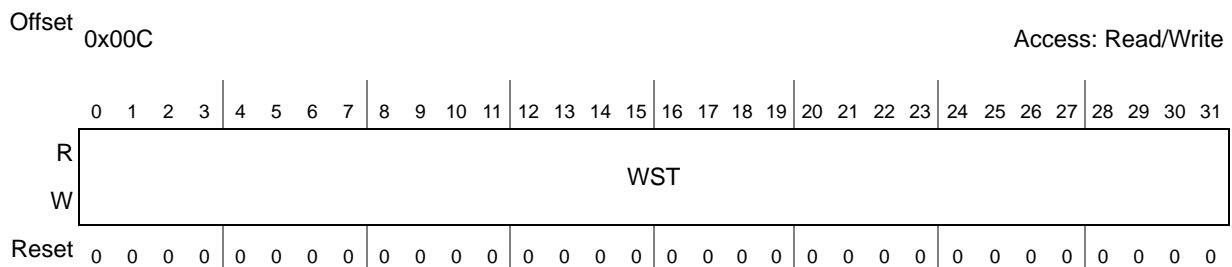
Default counter value (SWT\_TO\_RST) is 1280 (0x00000500 hexadecimal) which correspond to around 10 ms with a 128 kHz clock.

**Table 514. SWT\_TO Register field descriptions**

Field	Description
WTO	Watchdog time-out period in clock cycles. An internal 32-bit down counter is loaded with this value or 0x100 which ever is greater when the service sequence is written or when the SWT is enabled.

**33.5.2.4 SWT Window Register (SWT\_WN)**

The SWT Window (SWT\_WN) register contains the 32-bit window start value. This register is cleared on reset. This register is read only if either the SWT\_CR.HLK or SWT\_CR.SLK bits are set.



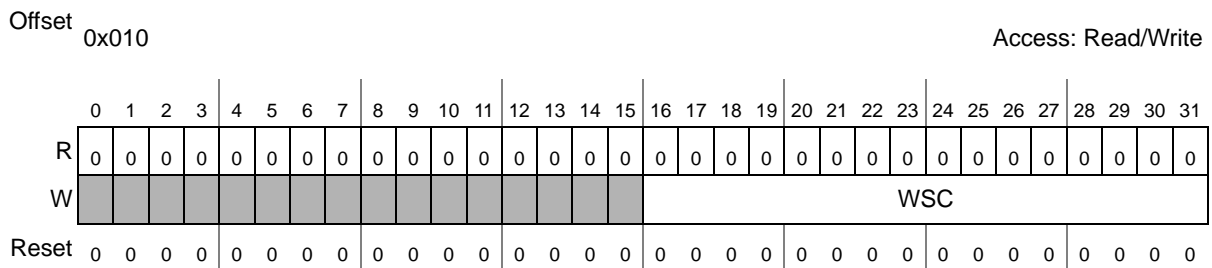
**Figure 546. SWT Window Register (SWT\_WN)**

**Table 515. SWT\_WN Register field descriptions**

Field	Description
WST	Window start value. When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

**33.5.2.5 SWT Service Register (SWT\_SR)**

The SWT Time-Out (SWT\_SR) service register is the target for service sequence writes used to reset the watchdog timer.



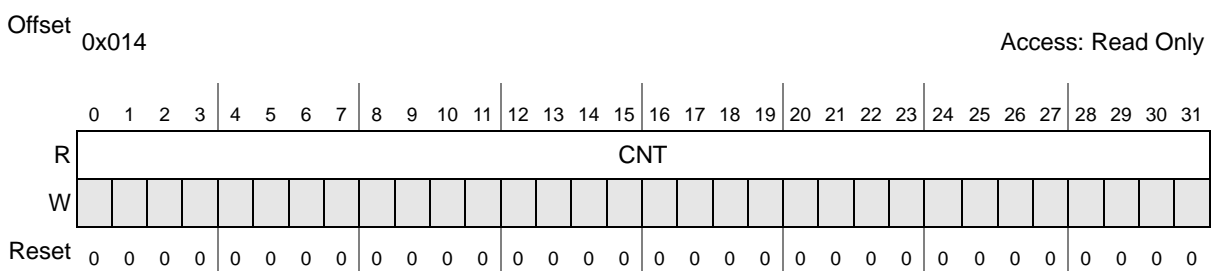
**Figure 547. SWT Service Register (SWT\_SR)**

**Table 516. SWT\_SR field descriptions**

Field	Description
WSC	Watchdog Service Code. This field is used to service the watchdog and to clear the soft lock bit (SWT_CR.SLK). To service the watchdog, the value 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_CR.SLKSWT_CR.), the value 0xC520 followed by 0xD928 is written to the WSC field.

**33.5.2.6 SWT Counter Output Register (SWT\_CO)**

The SWT Counter Output (SWT\_CO) register is a read only register that shows the value of the internal down counter when the SWT is disabled.



**Figure 548. SWT Counter Output Register (SWT\_CO)**

Table 517. SWT\_CO field descriptions

Field	Description
CNT	Watchdog Count. When the watchdog is disabled (SWT_CR.WEN=0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for up to six system plus eight counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter.

### 33.6 Functional description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a control register (SWT\_CR), an interrupt register (SWT\_IR), time-out register (SWT\_TO), a window register (SWT\_WN), a service register (SWT\_SR) and a counter output register (SWT\_CO).

The SWT\_CR includes bits to enable the timer, set configuration options and lock configuration of the module. The watchdog is enabled by setting the SWT\_CR.WEN bit. The reset value of the SWT\_CR.WEN bit is device specific<sup>1</sup> (enabled). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG\_EN) is '0'. If the reset value of this bit is 1, the watchdog starts operation automatically after reset is released. Some devices can be configured to clear this bit automatically during the boot process.

The SWT\_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x100 in which case the time-out period is set to 0x100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service sequence is written. The SWT\_CR.CSL bit selects which clock (system or oscillator) is used to drive the down counter. The reset value of the SWT\_TO register is device-specific as described previously.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT\_CR, SWT\_TO and SWT\_WN registers are read only. The hard lock is enabled by setting the SWT\_CR.HLK bit which can only be cleared by a reset. The soft lock is enabled by setting the SWT\_CR.SLK bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT\_SR.WSC field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT\_CR.WEN bit to be set.

When enabled, the SWT requires periodic execution of the watchdog servicing sequence. The service sequence is a write of 0xA602 followed by a write of 0xB480 to the SWT\_SR.WSC field. Writing the service sequence loads the internal down counter with the time-out period. There is no timing requirement between the two writes. The service sequence logic ignores unlock sequence writes and recognizes the 0xA602, 0xB480 sequence regardless of previous writes. Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require up to three system plus seven counter clock cycles.

If window mode is enabled (SWT\_CR.WND bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT\_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT\_CR.RIA bit. For example, if the SWT\_TO register is set to 5000 and SWT\_WN register is set to 1000 then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window to open due to synchronization logic in the watchdog design. This delay could be up to three system plus four counter clock cycles.

The interrupt then reset bit (SWT\_CR.ITR) controls the action taken when a time-out occurs. If the SWT\_CR.ITR bit is not set, a reset is generated immediately on a time-out. If the SWT\_CR.ITR bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT\_IR.TIF). The interrupt request is cleared by writing a one to the SWT\_IR.TIF bit.

The SWT\_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for up to six system plus eight counter clock cycles.

The SWT\_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT\_CR.WEN cleared) and the value of the SWT\_CO read to determine if the internal down counter is working properly.

*Note: Watchdog is disabled at the start of BAM execution. In the case of an unexpected issue during BAM execution, the CPU may be stalled and an external reset needs to be generated to recover.*

## 34 Error Correction Status Module (ECSM)

### 34.1 Introduction

The Error Correction Status Module (ECSM) provides a myriad of miscellaneous control functions for the device including program-visible information about configuration and revision levels, a reset status register, and information on memory errors reported by error-correcting codes.

### 34.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of miscellaneous control functions for the device.

### 34.3 Features

The ECSM includes these features:

- Program-visible information on the device configuration and revision
- Registers for capturing information on memory errors due to error-correction codes
- Registers to specify the generation of single- and double-bit memory data inversions for test purposes to check ECC protection
- Configuration for additional SRAM WS for system frequency above 64 + 4% MHz

### 34.4 Memory map and register description

This section details the programming model for the Error Correction Status Module. This is a 128-byte space mapped to the region serviced by an IPS bus controller.

#### 34.4.1 Memory map

The Error Correction Status Module does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

[Table 518](#) shows the ECSM’s memory map.

**Table 518. ECSM memory map**

Base address: 0xFFF4_0000		
Address offset	Register	Location
0x00	Processor Core Type Register (PCT)	<a href="#">on page 954</a>
0x02	SoC-Defined Platform Revision Register (REV)	<a href="#">on page 954</a>
0x04	Reserved	
0x08	IPS On-Platform Module Configuration Register (IOPMC)	<a href="#">on page 954</a>
0x0C–0x12	Reserved	



Table 518. ECSM memory map (continued)

Base address: 0xFFF4_0000		
Address offset	Register	Location
0x13	Miscellaneous Wakeup Control Register (MWCR)	<a href="#">on page 955</a>
0x14–0x1E	Reserved	
0x1F	Miscellaneous Interrupt Register (MIR)	<a href="#">on page 957</a>
0x20–0x23	Reserved	
0x24	Miscellaneous User-Defined Control Register (MUDCR)	<a href="#">on page 958</a>
0x28–0x42	Reserved	
0x43	ECC Configuration Register (ECR)	<a href="#">on page 959</a>
0x44–0x46	Reserved	
0x47	ECC Status Register (ESR)	<a href="#">on page 961</a>
0x48–0x49	Reserved	
0x4A	ECC Error Generation Register (EEGR)	<a href="#">on page 963</a>
0x4C–0x4F	Reserved	
0x50	Platform Flash ECC Address Register (PFEAR)	<a href="#">on page 965</a>
0x54–0x55	Reserved	
0x56	Platform Flash ECC Master Number Register (PFEMR)	<a href="#">on page 966</a>
0x57	Platform Flash ECC Attributes Register (PFEAT)	<a href="#">on page 967</a>
0x58–0x5B	Reserved	
0x5C	Platform Flash ECC Data Register (PFEDR)	<a href="#">on page 967</a>
0x60	Platform RAM ECC Address Register (PREAR)	<a href="#">on page 968</a>
0x64	Reserved	
0x65	Platform RAM ECC Syndrome Register (PRESR)	<a href="#">on page 969</a>
0x66	Platform RAM ECC Master Number Register (PREMR)	<a href="#">on page 971</a>
0x67	Platform RAM ECC Attributes Register (PREAT)	<a href="#">on page 971</a>
0x68–0x6B	Reserved	
0x6C	Platform RAM ECC Data Register (PREDR)	<a href="#">on page 972</a>

### 34.4.2 Register description

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, writes to the programming model must match the size of the register, e.g., an n-bit register only supports n-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

### 34.4.2.1 Processor Core Type Register (PCT)

The PCT is a 16-bit read-only register specifying the architecture of the processor core in the device. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Offset: 0x00 Access: Read

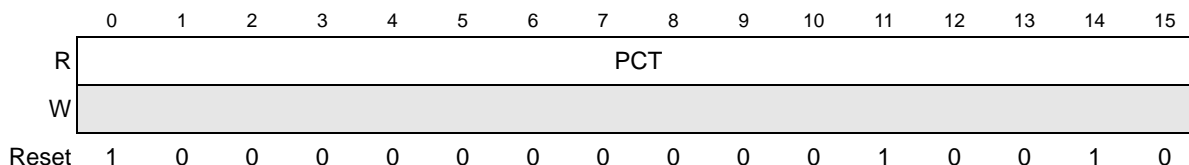


Figure 549. Processor Core Type Register (PCT)

Table 519. PCT field descriptions

Field	Description
PCT	Processor Core Type

### 34.4.2.2 SoC-Defined Platform Revision Register (REV)

The REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Offset: 0x02 Access: Read

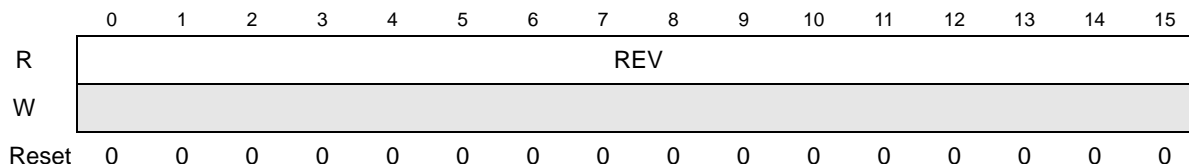


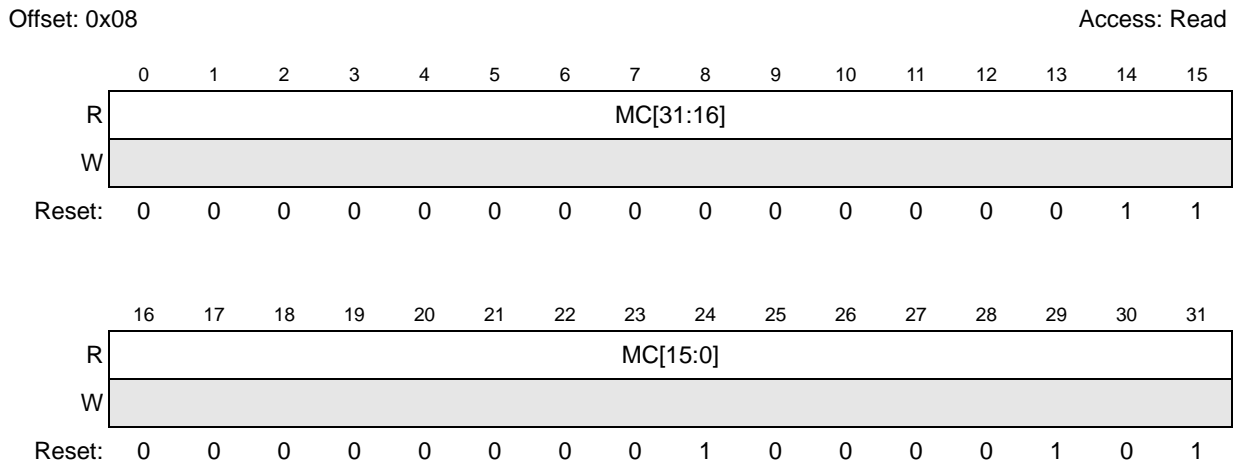
Figure 550. SoC-Defined Platform Revision Register (REV)

Table 520. REV field descriptions

Field	Description
REV	Revision The REV field is specified by an input signal to define a software-visible revision number.

### 34.4.2.3 IPS On-Platform Module Configuration Register (IOPMC)

The IOPMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order IPS peripheral modules connected to the primary IPI slave bus controller. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.



**Figure 551. IPS On-Platform Module Configuration Register (IOPMC)**

**Table 521. IOPMC field descriptions**

Field	Description
MC	IPS Module Configuration MC[n] = 0 if an IPS module connection to decoded slot “n” is absent MC[n] = 1 if an IPS module connection to decoded slot “n” is present

**34.4.2.4 Miscellaneous Wakeup Control Register (MWCR)**

Implementation of low-power sleep modes and exit from these modes via an interrupt require communication between the ECSM, the interrupt controller and off-platform external logic typically associated with phase-locked loop clock generation circuitry. The Miscellaneous Wakeup Control Register (MWCR) provides an 8-bit register controlling entry into these types of low-power modes as well as definition of the interrupt level needed to exit the mode.

The following sequence of operations is generally needed to enable this functionality. Note that the exact details are likely to be system-specific.

1. The processor core loads the appropriate data value into the MWCR, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor ceases execution. The exact mechanism varies by processor core. In some cases, a processor-is-stopped status is signaled to the ECSM and off-platform external logic. This assertion, if properly enabled by MWCR[ENBWCR], causes the ECSM output signal “enter\_low\_power\_mode” to be set. This, in turn, causes the selected off-platform external, low-power mode, as specified by MWCR[LPMD], to be entered, and the appropriate clock signals disabled. In most implementations, there are multiple low-power modes, where the exact clocks to be disabled vary across the different modes.
3. After entering the low-power mode, the interrupt controller enables a special combinational logic path which evaluates all unmasked interrupt requests. The device

- remains in this mode until an event which generates an unmasked interrupt request with a priority level greater than the value programmed in the MWCR[PRILVL] occurs.
4. Once the appropriately-high interrupt request level arrives, the interrupt controller signals its presence, and the ECSM responds by asserting an “exit\_low\_power\_mode” signal.
  5. The off-platform external logic senses the assertion of the “exit” signal, and re-enables the appropriate clock signals.
  6. With the processor core clocks enabled, the core handles the pending interrupt request.

Offset: 0x13

Access: Read/write

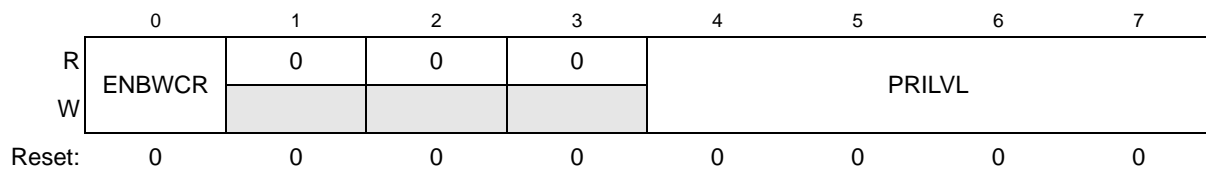


Figure 552. Miscellaneous Wakeup Control (MWCR) Register

Table 522. MWCR field descriptions

Field	Description
ENBWCR	Enable WCR 0 MWCR is disabled. 1 MWCR is enabled.
PRILVL	Interrupt Priority Level The interrupt priority level is a core-specific definition. It specifies the interrupt priority level needed to exit the low-power mode. Specifically, an unmasked interrupt request of a priority level greater than the PRILVL value is required to exit the mode.  Certain interrupt controller implementations include logic associated with this priority level that restricts the data value contained in this field to a [0, maximum - 1] range. See the specific interrupt controller module for details.

### 34.4.2.5 Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with ECSM are collected in the MIR. This includes the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the MIR must be explicitly cleared. See [Figure 553](#) and [Table 523](#).

Offset: 0x1F

Access: Special

	0	1	2	3	4	5	6	7
R	FB0AI	FB0SI	FB1AI	FB1SI	0	0	0	0
W	1	1	1	1				
Reset:	0	0	0	0	0	0	0	0

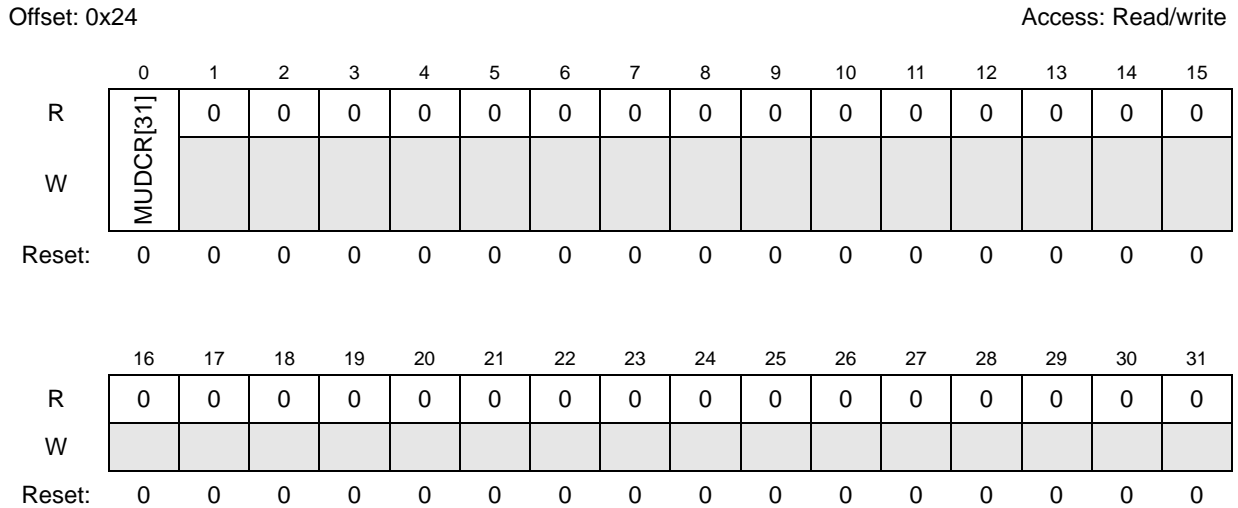
**Figure 553. Miscellaneous Interrupt (MIR) Register**

**Table 523. MIR field descriptions**

Field	Description
FB0AI	Flash Bank 0 Abort Interrupt 0 A flash bank 0 abort has not occurred. 1 A flash bank 0 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
FB0SI	Flash Bank 0 Stall Interrupt 0 A flash bank 0 stall has not occurred. 1 A flash bank 0 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
FB1AI	Flash Bank 1 Abort Interrupt 0 A flash bank 1 abort has not occurred. 1 A flash bank 1 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
FB1SI	Flash Bank 1 Stall Interrupt 0 A flash bank 1 stall has not occurred. 1 A flash bank 1 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.

### 34.4.2.6 Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions. It typically is used as configuration control for miscellaneous SoC-level modules. The contents of this register is simply output from the ECSM to other modules where the user-defined control functions are implemented.



**Figure 554. Miscellaneous User-Defined Control (MUDCR) Register**

**Table 524. MUDCR field descriptions**

Field	Description
MUDCR[31]	XBAR force_round_robin bit This bit is used to drive the force_round_robin bit of the XBAR. This will force the slaves into round robin mode of arbitration rather than fixed mode (unless a master is using priority elevation, which forces the design back into fixed mode regardless of this bit). By setting the hardware definition to ENABLE_ROUND_ROBIN_RESET, this bit will reset to 1. 1 XBAR is in round robin mode 0 XBAR is in fixed priority mode

### 34.4.2.7 ECC registers

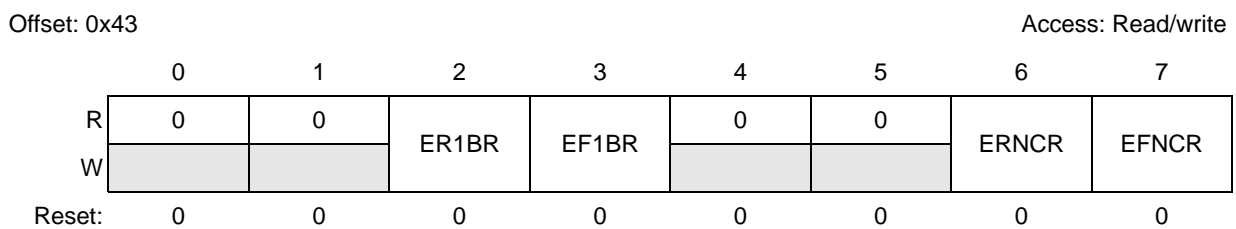
For designs including error-correcting code (ECC) implementations to improve the quality and reliability of memories, there are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers include:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Platform Flash ECC Address Register (PFEAR)
- Platform Flash ECC Master Number Register (PFEMR)
- Platform Flash ECC Attributes Register (PFEAT)
- Platform Flash ECC Data Register (PFEDR)
- Platform RAM ECC Address Register (PREAR)
- Platform RAM ECC Syndrome Register (PRESR)
- Platform RAM ECC Master Number Register (PREMR)
- Platform RAM ECC Attributes Register (PREAT)
- Platform RAM ECC Data Register (PREDR)

The details on the ECC registers are provided in the subsequent sections.

#### 34.4.2.7.1 ECC Configuration Register (ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches which are discarded due to a change-of-flow operation, and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) which may be useful for subsequent failure analysis.



**Figure 555. ECC Configuration (ECR) Register**

**Table 525. ECR field descriptions**

Field	Description
ER1BR	<p>Enable SRAM 1-bit Reporting</p> <p>This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit SRAM correction generates a ECSM ECC interrupt request as signalled by the assertion of ESR[R1BC]. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers.</p> <p>0 Reporting of single-bit SRAM corrections is disabled.                      1 Reporting of single-bit SRAM corrections is enabled.</p>
EF1BR	<p>Enable Flash 1-bit Reporting</p> <p>This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit flash correction generates a ECSM ECC interrupt request as signalled by the assertion of ESR[F1BC]. The address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers.</p> <p>0 Reporting of single-bit flash corrections is disabled.                      1 Reporting of single-bit flash corrections is enabled.</p>
ERNCR	<p>Enable SRAM Non-Correctable Reporting</p> <p>The occurrence of a non-correctable multi-bit SRAM error generates a ECSM ECC interrupt request as signalled by the assertion of ESR[RNCE]. The faulting address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers.</p> <p>0 Reporting of non-correctable SRAM errors is disabled.                      1 Reporting of non-correctable SRAM errors is enabled.</p>
EFNCR	<p>Enable Flash Non-Correctable Reporting</p> <p>The occurrence of a non-correctable multi-bit flash error generates a ECSM ECC interrupt request as signalled by the assertion of ESR[FNCE]. The faulting address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers.</p> <p>0 Reporting of non-correctable flash errors is disabled.                      1 Reporting of non-correctable flash errors is enabled.</p>



### 34.4.2.7.2 ECC Status Register (ESR)

The ECC Status Register is an 8-bit control register for signaling which types of properly-enabled ECC events have been detected. The ESR signals the last, properly-enabled memory event to be detected. ECC interrupt generation is separated into single-bit error detection/correction, uncorrectable error detection and the combination of the two as defined by the following boolean equations:

ECSM\_ECC1BIT\_IRQ

= ECR[ER1BR] & ESR[R1BC] // ram, 1-bit correction  
 | ECR[EF1BR] & ESR[F1BC] // flash, 1-bit correction

ECSM\_ECCRNCR\_IRQ

= ECR[ERNCR] & ESR[RNCE] // ram, noncorrectable error

ECSM\_ECCFNCR\_IRQ

= ECR[EFNCR] & ESR[FNCE] // flash, noncorrectable error

ECSM\_ECC2BIT\_IRQ

= ECSM\_ECCRNCR\_IRQ // ram, noncorrectable error  
 | ECSM\_ECCFNCR\_IRQ // flash, noncorrectable error

ECSM\_ECC\_IRQ

= ECSM\_ECC1BIT\_IRQ // 1-bit correction  
 | ECSM\_ECC2BIT\_IRQ // noncorrectable error

where the combination of a properly-enabled category in the ECR and the detection of the corresponding condition in the ESR produces the interrupt request.

The ECSM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of a properly-enabled ECC event. If there is a pending ECC interrupt and another properly-enabled ECC event occurs, the ECSM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

Offset: 0x47

Access: Read/write

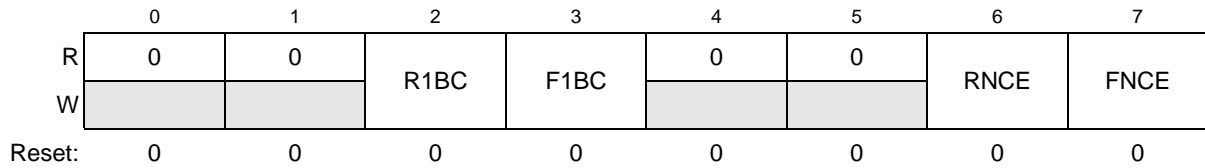


Figure 556. ECC Status Register (ESR)

Table 526. ESR field descriptions

Field	Description
R1BC	<p>SRAM 1-bit Correction</p> <p>This bit can only be set if ECR[EPR1BR] is asserted. The occurrence of a properly-enabled single-bit SRAM correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable single-bit SRAM correction has been detected.                      1 A reportable single-bit SRAM correction has been detected.</p>
F1BC	<p>Flash Memory 1-bit Correction</p> <p>This bit can only be set if ECR[EPF1BR] is asserted. The occurrence of a properly-enabled single-bit flash memory correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable single-bit flash memory correction has been detected.                      1 A reportable single-bit flash memory correction has been detected.</p>
RNCE	<p>SRAM Non-Correctable Error</p> <p>The occurrence of a properly-enabled non-correctable SRAM error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the PREAR, PRESR, PREMR, PREAT and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable non-correctable SRAM error has been detected.                      1 A reportable non-correctable SRAM error has been detected.</p>
FNCE	<p>Flash Memory Non-Correctable Error</p> <p>The occurrence of a properly-enabled non-correctable flash memory error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the PFEAR, PFEMR, PFEAT and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable non-correctable flash memory error has been detected.                      1 A reportable non-correctable flash memory error has been detected.</p>

In the event that multiple status flags are signaled simultaneously, ECSM records the event with the R1BC as highest priority, then F1BC, then RNCE, and finally FNCE.

### 34.4.2.7.3 ECC Error Generation Register (EEGR)

The ECC Error Generation Register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the SRAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for “injecting” errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the EEGR is associated with the SRAM, similar capabilities exist for the flash, that is, the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (SRAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

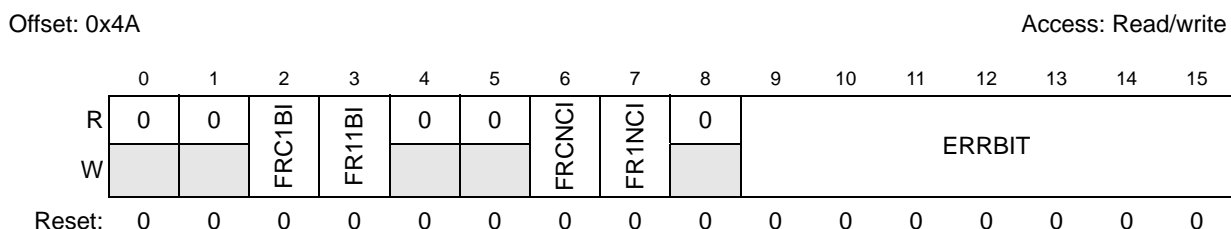


Figure 557. ECC Error Generation Register (EEGR)

Table 527. EEGR field descriptions

Field	Description
FRC1BI	<p>Force SRAM Continuous 1-bit Data Inversions</p> <p>The assertion of this bit forces the SRAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the SRAM.</p> <p>After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>This bit can only be set if the same SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted.</p> <p>0 No SRAM continuous 1-bit data inversions are generated.                      1 1-bit data inversions in the SRAM are continuously generated.</p>

**Table 527. EEGR field descriptions (continued)**

Field	Description
FR11BI	<p>Force SRAM One 1-bit Data Inversion The assertion of this bit forces the SRAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the SRAM.</p> <p>After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>This bit can only be set if the same SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted.</p> <p>0 No SRAM single 1-bit data inversion is generated. 1 One 1-bit data inversion in the SRAM is generated.</p>
FRCNCI	<p>Force SRAM Continuous Non-correctable Data Inversions The assertion of this bit forces the SRAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, continuously on every write operation.</p> <p>After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the SRAM.</p> <p>0 No SRAM continuous 2-bit data inversions are generated. 1 2-bit data inversions in the SRAM are continuously generated.</p>
FR1NCI	<p>Force SRAM One Non-correctable Data Inversions The assertion of this bit forces the SRAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, on the first write operation after this bit is set.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the SRAM.</p> <p>After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>0 No SRAM single 2-bit data inversions are generated. 1 One 2-bit data inversion in the SRAM is generated.</p>

**Table 527. EEGR field descriptions (continued)**

Field	Description
ERRBIT	<p>Error Bit Position</p> <p>The vector defines the bit position which is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted.</p> <p>The SRAM controller follows a vector bit ordering scheme where LSB = 0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the SRAM width. For example, consider a 32-bit SRAM implementation.</p> <p>The 32-bit ECC approach requires 7 code bits for a 32-bit word. For PRAM data width of 32 bits, the actual SRAM (32b data + 7b for ECC) = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined:</p> <p>if ERRBIT = 0, then SRAM[0] of the odd bank is inverted                      if ERRBIT = 1, then SRAM[1] of the odd bank is inverted                      ...                      if ERRBIT = 31, then SRAM[31] of the odd bank is inverted                      if ERRBIT = 64, then ECC Parity[0] of the odd bank is inverted                      if ERRBIT = 65, then ECC Parity[1] of the odd bank is inverted                      ...                      if ERRBIT = 70, then ECC Parity[6] of the odd bank is inverted</p> <p>For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.</p>

If an attempt to force a non-correctable inversion (by asserting EEGR[FRCNCI] or EEGR[FRC1NCI]) and EEGR[ERRBIT] equals 64, then no data inversion will be generated.

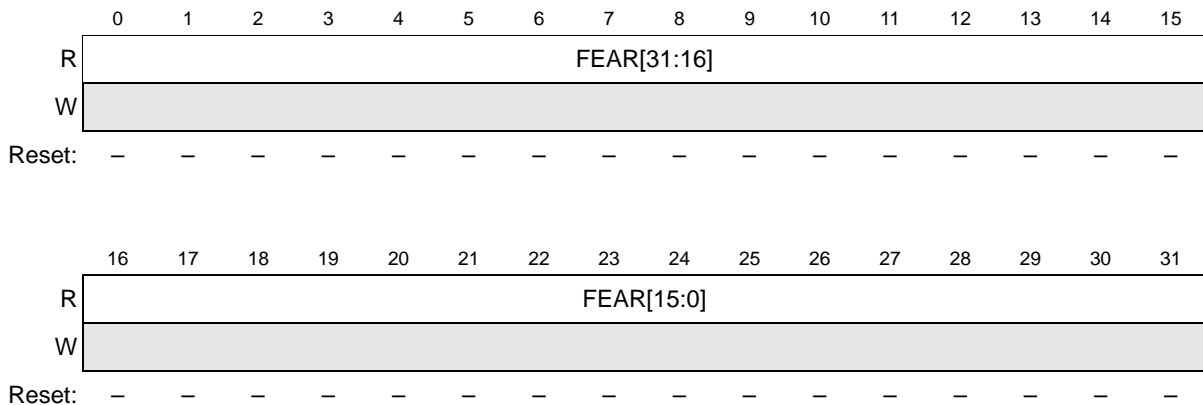
The only allowable values for the 4 control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in undefined behavior.

**34.4.2.7.4 Platform Flash ECC Address Register (PFEAR)**

The PFEAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Offset: 0x50 Access: Read



**Figure 558. Platform Flash ECC Address Register (PFEAR)**

**Table 528. PFEAR field descriptions**

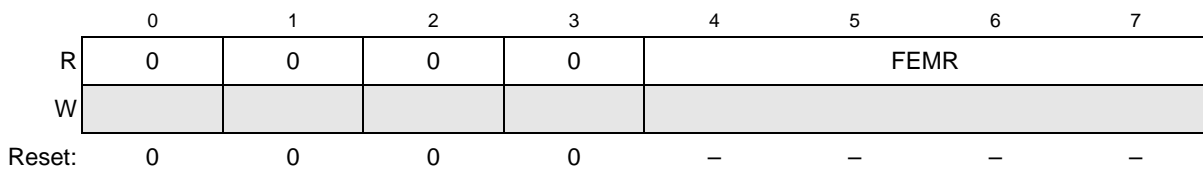
Field	Description
FEAR	Flash ECC Address Register This 32-bit register contains the faulting access address of the last, properly-enabled flash ECC event.

**34.4.2.7.5 Platform Flash ECC Master Number Register (PFEMR)**

The PFEMR is a 4-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Offset: 0x56 Access: Read



**Figure 559. Platform Flash ECC Master Number Register (PFEMR)**

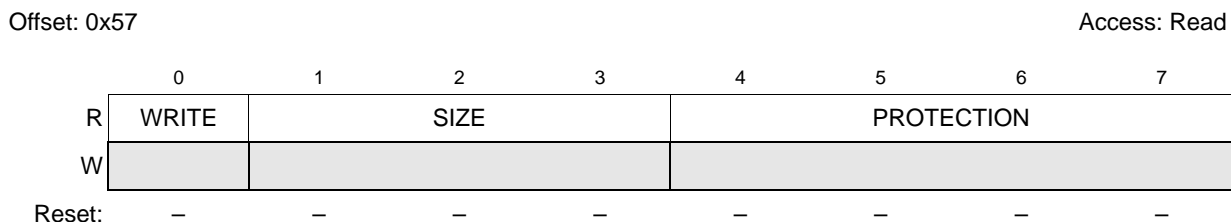
**Table 529. PFEMR field descriptions**

Field	Description
FEMR	Flash ECC Master Number Register This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly-enabled flash ECC event.

### 34.4.2.7.6 Platform Flash ECC Attributes Register (PFEAT)

The PFEAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.



**Figure 560. Platform Flash ECC Attributes Register (PFEAT)**

**Table 530. PFEAT field descriptions**

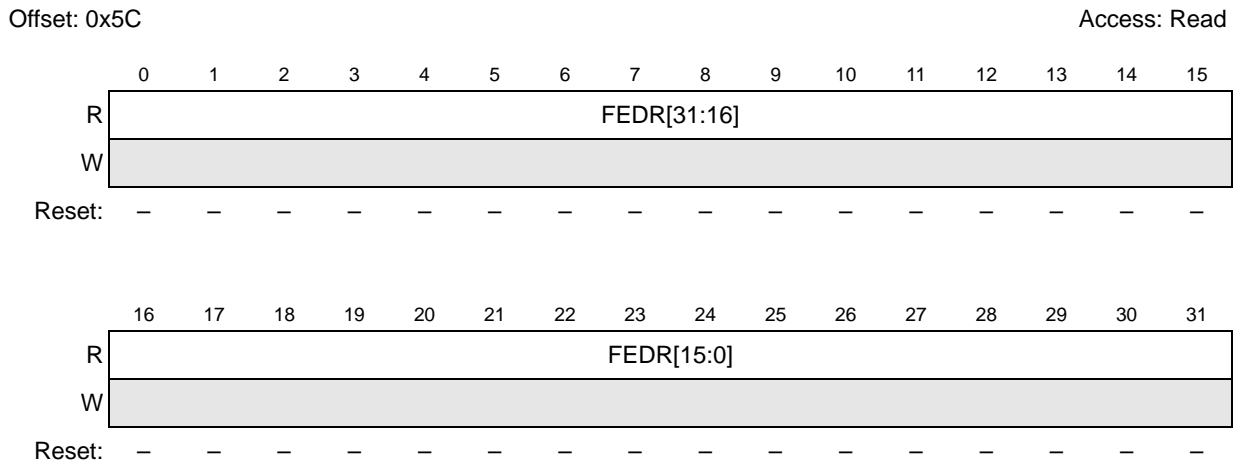
Field	Description
WRITE	AMBA-AHB HWRITE 0 AMBA-AHB read access 1 AMBA-AHB write access
SIZE	AMBA-AHB HSIZE[2:0] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1xx Reserved
PROTECTION	AMBA-AHB HPROT[3:0] Protection[3]: Cacheable0 = Non-cacheable, 1 = Cacheable Protection[2]: Bufferable0 = Non-bufferable, 1 = Bufferable Protection[1]: Mode0 = User mode, 1 = Supervisor mode Protection[0]: Type0 = I-Fetch, 1 = Data

### 34.4.2.7.7 Platform Flash ECC Data Register (PFEDR)

The PFEDR is a 32-bit register for capturing the data associated with the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.



**Figure 561. Platform Flash ECC Data Register (PFEDR)**

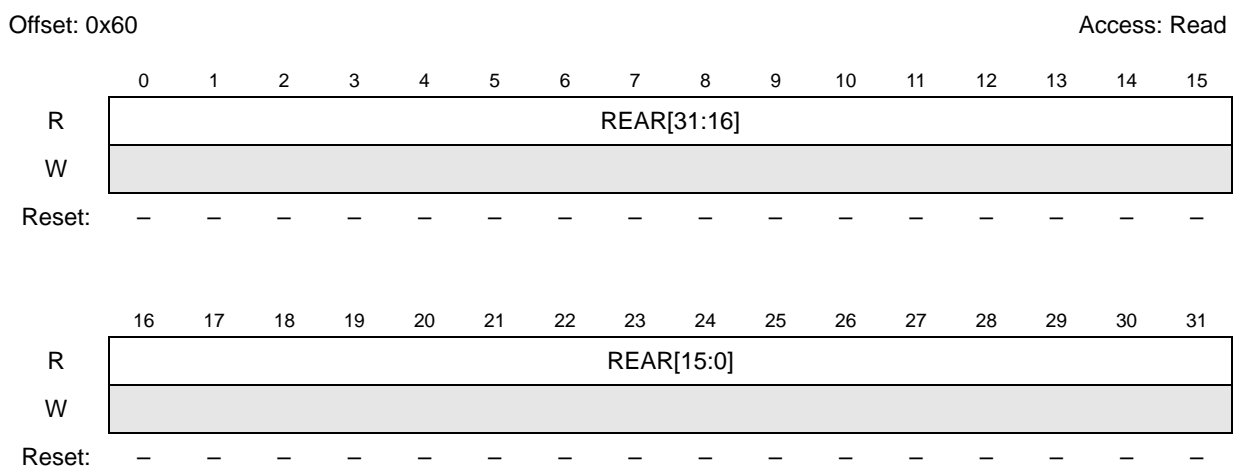
**Table 531. PFEDR field descriptions**

Field	Description
FEDR	Flash ECC Data Register This 32-bit register contains the data associated with the faulting access of the last, properly-enabled flash ECC event. The register contains the data value taken directly from the data bus.

**34.4.2.7.8 Platform RAM ECC Address Register (PREAR)**

The PREAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.



**Figure 562. Platform RAM ECC Address Register (PREAR)**



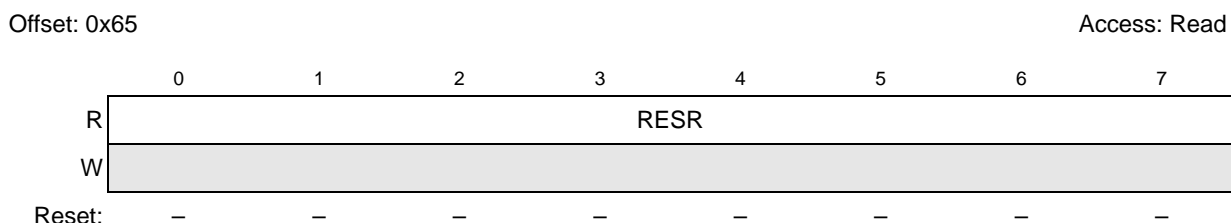
**Table 532. PREAR field descriptions**

Field	Description
REAR	SRAM ECC Address Register This 32-bit register contains the faulting access address of the last, properly-enabled SRAM ECC event.

**34.4.2.7.9 Platform RAM ECC Syndrome Register (PRESR)**

The PRESR is an 8-bit register for capturing the error syndrome of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.



**Figure 563. Platform RAM ECC Syndrome Register (PRESR)**

**Table 533. PRESR field descriptions**

Field	Description
RESR	SRAM ECC Syndrome Register This 8-bit syndrome field includes 6 bits of Hamming decoded parity plus an odd-parity bit for the entire 39-bit (32-bit data + 7 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error.  For correctable single-bit errors, the mapping shown in <a href="#">Table 534</a> associates the upper 7 bits of the syndrome with the data bit in error.

[Table 534](#) associates the upper 7 bits of the ECC syndrome with the exact data bit in error for single-bit correctable codewords. This table follows the bit vectoring notation where the LSB = 0. Note that the syndrome value of 0x01 implies no error condition but this value is not readable when the PRESR is read for the no error case.

**Table 534. RAM syndrome mapping for single-bit correctable errors**

PRESR[RESR]	Data bit in error
0x00	ECC ODD[0]
0x01	No error

**Table 534. RAM syndrome mapping for single-bit correctable errors (continued)**

PRESR[RESR]	Data bit in error
0x02	ECC ODD[1]
0x04	ECC ODD[2]
0x06	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x0a	DATA ODD BANK[30]
0x0c	DATA ODD BANK[29]
0x0e	DATA ODD BANK[28]
0x10	ECC ODD[4]
0x12	DATA ODD BANK[27]
0x14	DATA ODD BANK[26]
0x16	DATA ODD BANK[25]
0x18	DATA ODD BANK[24]
0x1a	DATA ODD BANK[23]
0x1c	DATA ODD BANK[22]
0x50	DATA ODD BANK[21]
0x20	ECC ODD[5]
0x22	DATA ODD BANK[20]
0x24	DATA ODD BANK[19]
0x26	DATA ODD BANK[18]
0x28	DATA ODD BANK[17]
0x2a	DATA ODD BANK[16]
0x2c	DATA ODD BANK[15]
0x58	DATA ODD BANK[14]
0x30	DATA ODD BANK[13]
0x32	DATA ODD BANK[12]
0x34	DATA ODD BANK[11]
0x64	DATA ODD BANK[10]
0x38	DATA ODD BANK[9]
0x62	DATA ODD BANK[8]
0x70	DATA ODD BANK[7]
0x60	DATA ODD BANK[6]
0x40	ECC ODD[6]
0x42	DATA ODD BANK[5]
0x44	DATA ODD BANK[4]
0x46	DATA ODD BANK[3]

**Table 534. RAM syndrome mapping for single-bit correctable errors (continued)**

PRESR[RESR]	Data bit in error
0x48	DATA ODD BANK[2]
0x4a	DATA ODD BANK[1]
0x4c	DATA ODD BANK[0]
0x03,0x05.....0x4d	Multiple bit error
> 0x4d	Multiple bit error

**34.4.2.7.10 Platform RAM ECC Master Number Register (PREMR)**

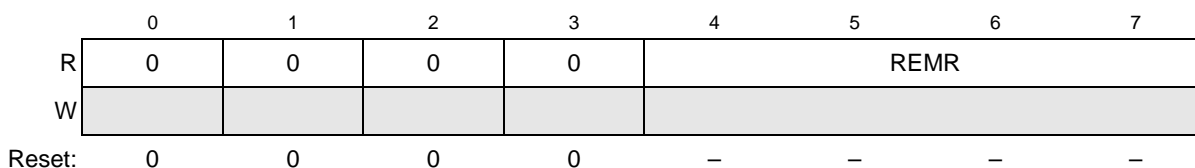
The PREMR is a 4-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

See the XBAR chapter of this reference manual for a listing of XBAR bus master numbers.

This register can only be read from the IPS programming model; any attempted write is ignored.

Offset: 0x66

Access: Read



**Figure 564. Platform RAM ECC Master Number Register (PREMR)**

**Table 535. PREMR field descriptions**

Field	Description
PREMR	SRAM ECC Master Number Register This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly-enabled SRAM ECC event. See the XBAR chapter of this reference manual for a listing of XBAR bus master numbers.

**34.4.2.7.11 Platform RAM ECC Attributes Register (PREAT)**

The PREAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

Offset: 0x67

Access: Read

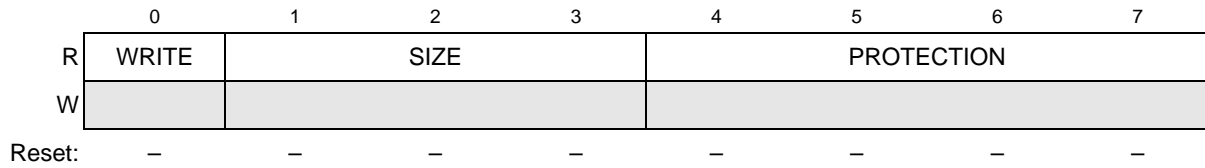


Figure 565. Platform RAM ECC Attributes Register (PREAT)

Table 536. PREAT field descriptions

Field	Description
WRITE	XBAR HWRITE 0 XBAR read access 1 XBAR write access
SIZE	XBAR HSIZE[2:0] 000 8-bit XBAR access 001 16-bit XBAR access 010 32-bit XBAR access 1xx Reserved
PROTECTION	XBAR HPROT[3:0] Protection[3]: Cacheable 0 = Non-cacheable, 1 = Cacheable Protection[2]: Bufferable 0 = Non-bufferable, 1 = Bufferable Protection[1]: Mode 0 = User mode, 1 = Supervisor mode Protection[0]: Type 0 = I-Fetch, 1 = Data

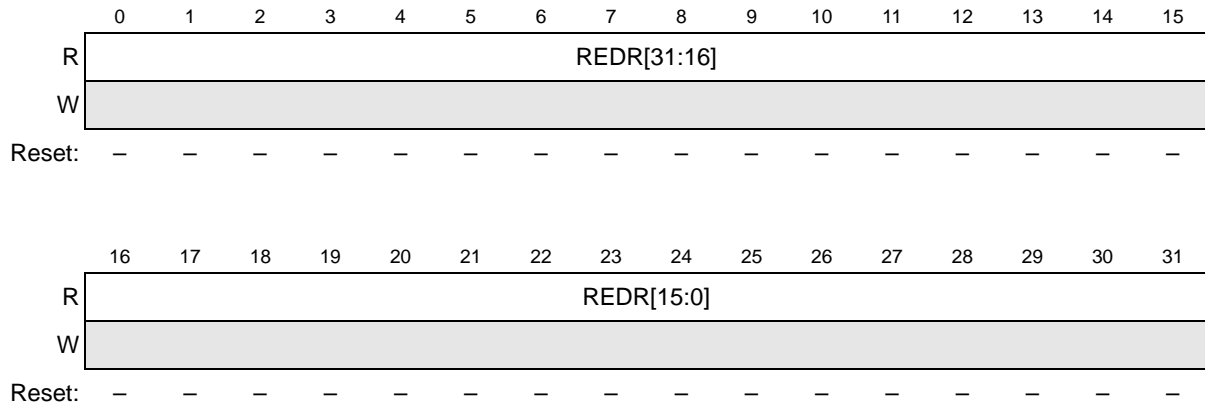
34.4.2.7.12 Platform RAM ECC Data Register (PREDR)

The PREDR is a 32-bit register for capturing the data associated with the last, properly-enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

Offset: 0x6C

Access: Read



**Figure 566. Platform RAM ECC Data Register (PREDR)**

**Table 537. PREDR field descriptions**

Field	Description
REDR	SRAM ECC Data Register This 32-bit register contains the data associated with the faulting access of the last, properly-enabled SRAM ECC event. The register contains the data value taken directly from the data bus.

### 34.4.3 Register protection

Logic exists which restricts accesses to INTC, ECSM, MPU, STM and SWT to supervisor mode only. Accesses in User mode are not possible.

## 35 IEEE 1149.1 Test Access Port Controller (JTAGC)

### 35.1 Introduction

The JTAG port of the device consists of three inputs and one output. These pins include test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

### 35.2 Block diagram

Figure 567 is a block diagram of the JTAG Controller (JTAGC) block.

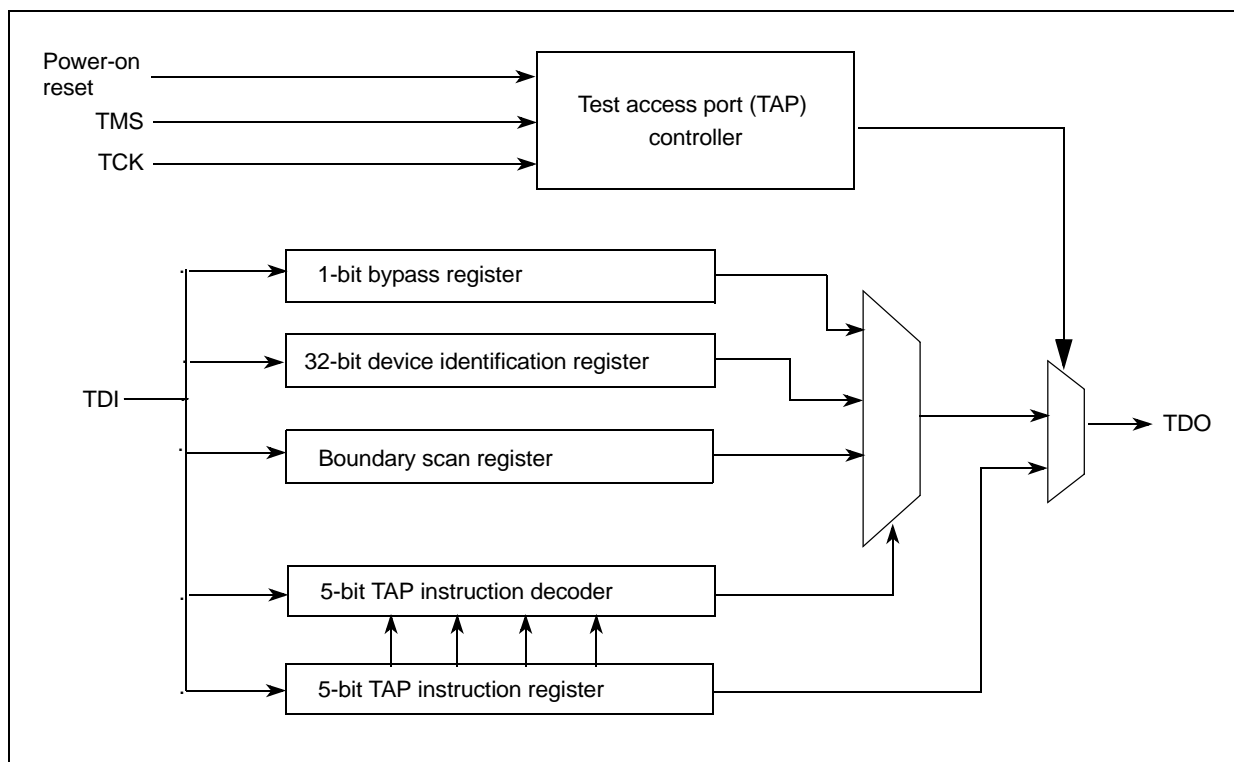


Figure 567. JTAG Controller Block Diagram

### 35.3 Overview

The JTAGC provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in TEST mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. In addition, instructions can be executed that allow the Test Access Port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

## 35.4 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface
- 4 pins (TDI, TMS, TCK, and TDO)—Refer to [Section 35.6: External signal description](#)
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions
- 2 test data registers:
  - Bypass register
  - Device identification register
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry

## 35.5 Modes of operation

The JTAGC uses a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined TEST modes are supported, as well as a bypass mode.

### 35.5.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST.

### 35.5.2 IEEE 1149.1-2001 defined test modes

The JTAGC supports several IEEE 1149.1-2001 defined TEST modes. The TEST mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data registers that can operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is external to JTAGC but can be accessed by JTAGC TAP through EXTEST, SAMPLE, SAMPLE/PRELOAD instructions. The functionality of each TEST mode is explained in more detail in [Section 35.8.4: JTAGC instructions](#).

#### 35.5.2.1 Bypass Mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register is used to provide a minimum-length serial path to shift data between TDI and TDO.

### 35.5.2.2 TAP sharing mode

There are three selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC) and PLATFORM. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS\_AUX\_TAP\_NPC, ACCESS\_AUX\_TAP\_ONCE, ACCESS\_AUX\_TAP\_TCU. Instruction opcodes for each instruction are shown in [Table 540](#).

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

For more information on the TAP controllers refer to the Nexus port controller chapter of the reference manual.

## 35.6 External signal description

The JTAGC consists of four signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in [Table 538](#):

**Table 538. JTAG signal properties**

Name	I/O	Function	Reset State
TCK	I	Test clock	Pull Up
TDI	I	Test data in	Pull Up
TDO	O	Test data out	High Z
TMS	I	Test mode select	Pull Up

The JTAGC pins are shared with GPIO. TDO at reset is a input pad and output direction control from JTAGC. Once TAP enters shift-ir or shift-dr then output direction control from JTAGC which allows the value to see on pad. It is up to the user to configure them as GPIOs accordingly, .

## 35.7 Memory map and register description

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

### 35.7.1 Instruction Register

The JTAGC uses a 5-bit instruction register as shown in [Table 568](#). The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the update-IR and test-logic-



reset TAP controller states. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register’s read value when the TAP controller is sequenced into the Shift-IR state.

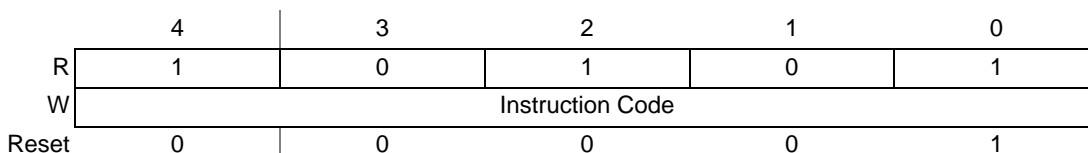


Figure 568. 5-bit Instruction Register

### 35.7.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

### 35.7.3 Device Identification Register

The device identification register, shown in [Figure 569](#), allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

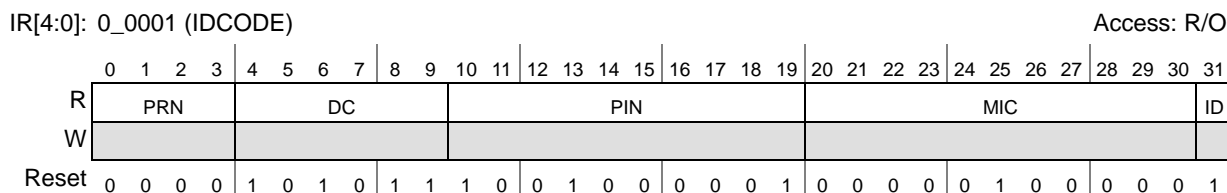


Figure 569. Device Identification Register

Table 539. Device Identification Register Field Descriptions

Field	Description
0–3 PRN	Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module.
4–9 DC	Design center. For the SPC560B54/6x this value is 0x2B.
10–19 PIN	Part identification number. Contains the part number of the device. For the SPC560B54/6x, this value is 0x243.

**Table 539. Device Identification Register Field Descriptions**

Field	Description
20–30 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for STMicroelectronics, 0x20.
31 ID	IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

### 35.7.4 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in [Section 35.8.5: Boundary Scan](#). The size of the boundary scan register is 464 bits.

## 35.8 Functional Description

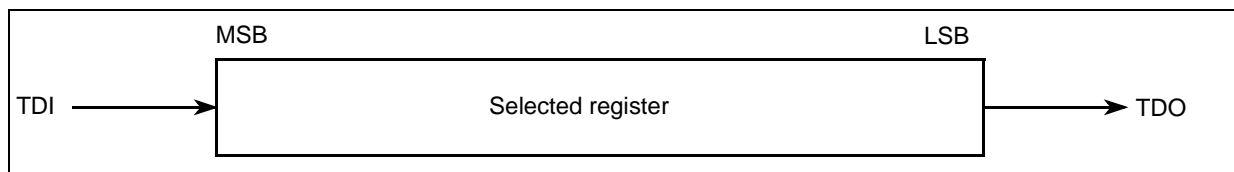
### 35.8.1 JTAGC Reset Configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

### 35.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. For more detail on TAP sharing via JTAGC instructions refer to [Section 35.8.4.2: ACCESS\\_AUX\\_TAP\\_x instructions](#).

Data is shifted between TDI and TDO through the selected register starting with the least significant bit, as illustrated in [Figure 570](#). This applies for the instruction register, test data registers, and the bypass register.



**Figure 570. Shifting data through a register**

### 35.8.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. [Figure 571](#) shows the machine’s states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal.

As [Figure 571](#) shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.

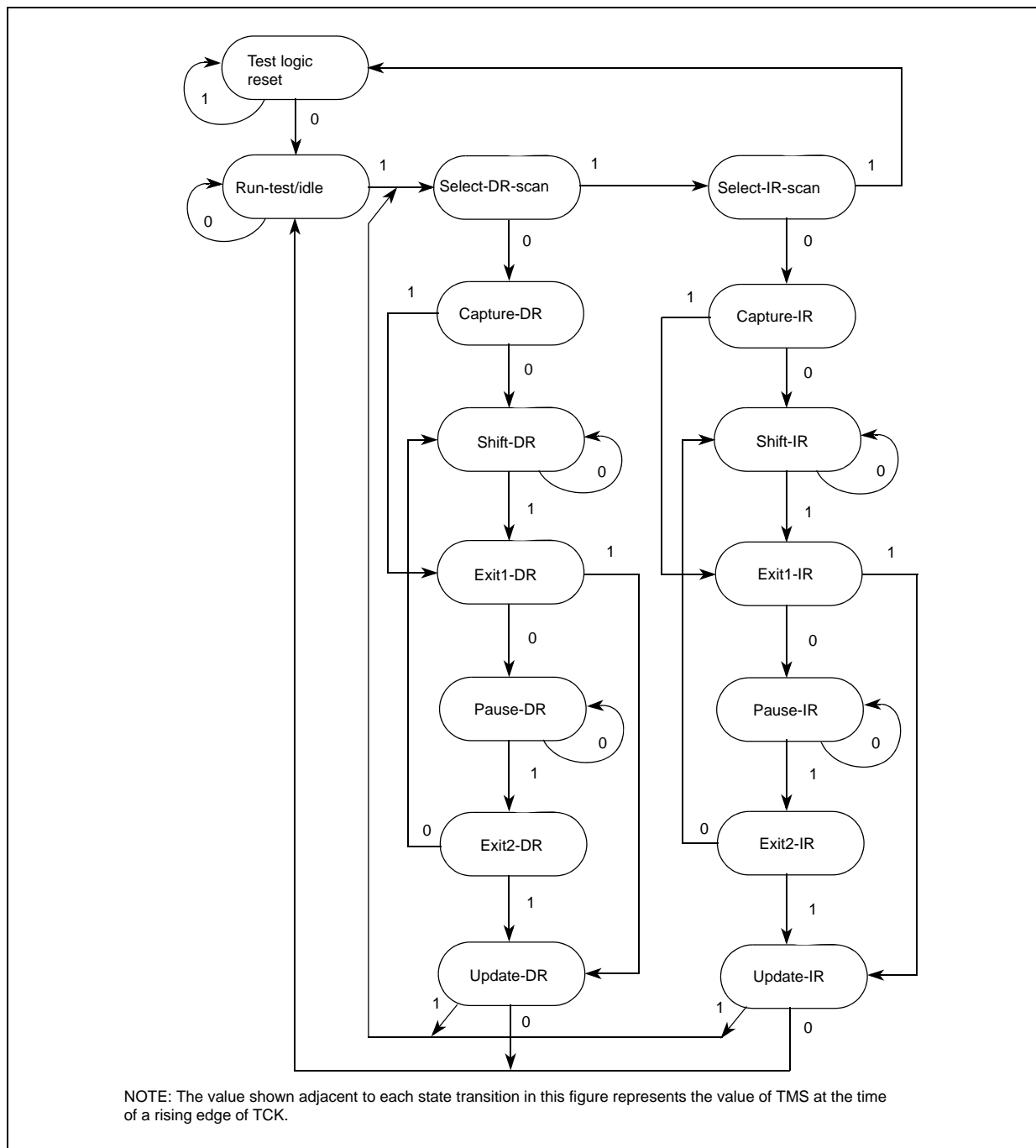


Figure 571. IEEE 1149.1-2001 TAP controller finite state machine

35.8.3.1 Selecting an IEEE 1149.1-2001 register

Access to the JTAGC data registers is done by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path is used to read or write the register data by shifting in the data (LSB first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after fetching the required number of bits.

### 35.8.4 JTAGC instructions

This section gives an overview of each instruction, refer to the IEEE 1149.1-2001 standard for more details.

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in [Table 540](#).

**Table 540. JTAG Instructions**

Instruction	Code[4:0]	Instruction Summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_TCU	11011	Grants the TCU ownership of the TAP
ACCESS_AUX_TAP_ONCE	10001	Grants the PLATFORM ownership of the TAP
ACCESS_AUX_TAP_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
Reserved	10010	—
BYPASS	11111	Selects bypass register for data operations
Factory Debug Reserved <sup>(1)</sup>	00101 00110 01010	Intended for factory debug only
Reserved <sup>(2)</sup>	All other codes	Decoded to select bypass register

1. Intended for factory debug, and not customer use
2. STMicroelectronics reserves the right to change the decoding of reserved instruction codes

#### 35.8.4.1 BYPASS instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

#### 35.8.4.2 ACCESS\_AUX\_TAP\_x instructions

The ACCESS\_AUX\_TAP\_x instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the



selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

#### 35.8.4.3 EXTEST — External Test Instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

#### 35.8.4.4 IDCODE instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

#### 35.8.4.5 SAMPLE instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the update-DR state. Both the data capture and the shift operation are transparent to system operation.

During the SAMPLE instruction, the following pad status is enforced:

- Weak pull is disabled (independent from PCRx[WPE])
- Analog switch is disabled (independent of PCRx[APC])
- Slew rate control is forced to the slowest configuration (independent from PCRx[SRC[1]])

#### 35.8.4.6 SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- The SAMPLE part of the instruction samples the system data and control signals on the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising-edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- The PRELOAD part of the instruction initializes the boundary scan register cells before selecting the EXTEST instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the update-DR state. The data is applied to the external output pins by the EXTEST instruction. System operation is not affected.

During the SAMPLE/PRELOAD instruction, the following pad status is enforced:

- Weak pull is disabled (independent from PCRx[WPE])
- Analog switch is disabled (independent of PCRx[APC])
- Slew rate control is forced to the slowest configuration (independent from PCRx[SRC[1]])

### 35.8.5 Boundary Scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

## 35.9 e200z0 OnCE controller

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug features, as well as providing access to the Nexus2+ configuration registers. A complete discussion of the e200z0 OnCE debug features is available in the *e200z0 Reference Manual*.

### 35.9.1 e200z0 OnCE Controller Block Diagram

[Figure 572](#) is a block diagram of the e200z0 OnCE block.

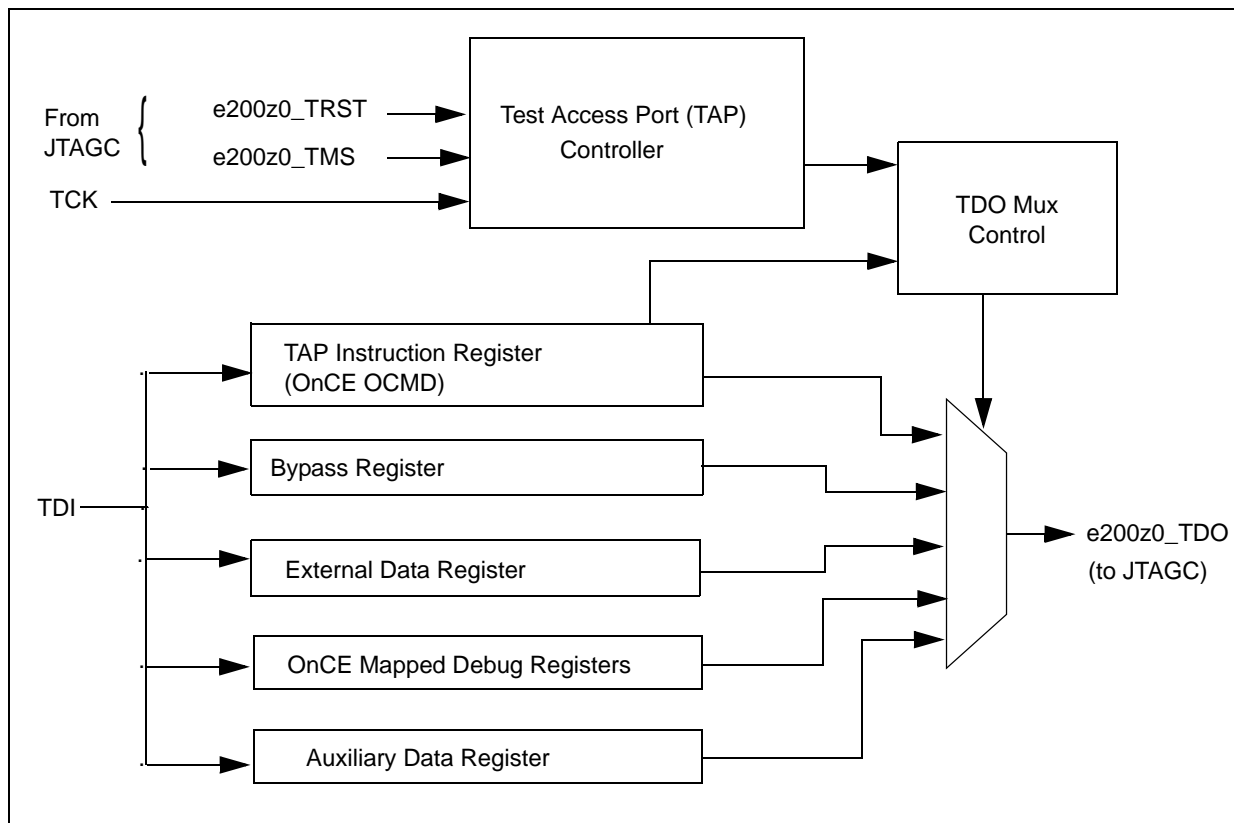


Figure 572. e200z0 OnCE Block Diagram

## 35.9.2 e200z0 OnCE Controller Functional Description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described below.

### 35.9.2.1 Enabling the TAP Controller

To access the e200z0 OnCE controller, the proper JTAGC instruction needs to be loaded in the JTAGC instruction register, as discussed in [Section 35.5.2.2: TAP sharing mode](#).

## 35.9.3 e200z0 OnCE Controller Register Description

Most e200z0 OnCE debug registers are fully documented in the *e200z0 Reference Manual*.

### 35.9.3.1 OnCE Command Register (OCMD)

The OnCE command register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0 OnCE Decoder. The OCMD is shown in [Table 573](#). The OCMD is updated when the TAP controller enters the update-IR state. It contains fields for controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as

such, the update-DR state must be transitioned through in order for an access to occur. In addition, the update-DR state must also be transitioned through in order for the single-step and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

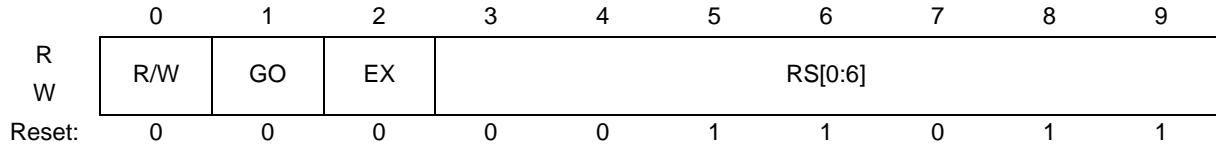


Figure 573. OnCE Command Register (OCMD)

Table 541. e200z0 OnCE Register Addressing

RS[0:6]	Register Selected
000 0000 000 0001	Reserved
000 0010	JTAG ID (read-only)
000 0011 – 000 1111	Reserved
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011 – 001 1111	Reserved
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110	Data Value Compare 1 (DVC1)
010 0111	Data Value Compare 2 (DVC2)
010 1000 – 010 1111	Reserved
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug Control Register 1 (DBCR1)
011 0011	Debug Control Register 2 (DBCR2)
011 0100 – 101 1111	Reserved (do not access)
110 1111	Shared Nexus Control Register (SNC) (only available on the e200z0 core)
111 0000 – 111 1001	General Purpose Register Selects [0:9]
111 1010 – 111 1011	Reserved
111 1100	Nexus2+ Access



Table 541. e200z0 OnCE Register Addressing (continued)

RS[0:6]	Register Selected
111 1101	LSRL Select (factory test use only)
111 1110	Enable_OnCE
111 1111	Bypass

## 35.10 Initialization/application information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Place the JTAGC in reset through TAP controller state machine transitions controlled by TMS
2. Load the appropriate instruction for the test or action to be performed.

## 36 Nexus Development Interface (NDI)

### 36.1 Introduction

SPC560B54/6x contains single Nexus client (Nexus2+) that communicates over an auxiliary port interface compliant to IEEE-ISTO 5001-2003 standard and a JTAG port compliant with IEEE 1149.1 standard. This JTAG port is also shared with the OnCE/Nexus1 unit.

Nexus 2+ module provides real-time development capabilities for e200z0h core processor in compliance with Class 2 of the IEEE-ISTO 5001-2003 standard.

- The auxiliary port comprises 7 output pins and 1 input pin. The output pins include 1 message clock out (MCKO) pin, 4 message data out (MDO) pins, 1 message start/end out (MSEO) pins, and 1 event out (EVTO) pin. Event in (EVTI) is the only input pin for the auxiliary port.
- The JTAG port consists of three inputs and one output. These pins include test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAG controller (JTAGC). See [Section 36.4: Memory map and registers](#), for the JTAGC opcodes to access the different Nexus clients.

### 36.2 Block diagram

[Figure 574](#) shows a functional block diagram of the NDI. [Figure 575](#) shows an implementation block diagram of the NDI, which shows how the individual Nexus blocks are combined to form the NDI.

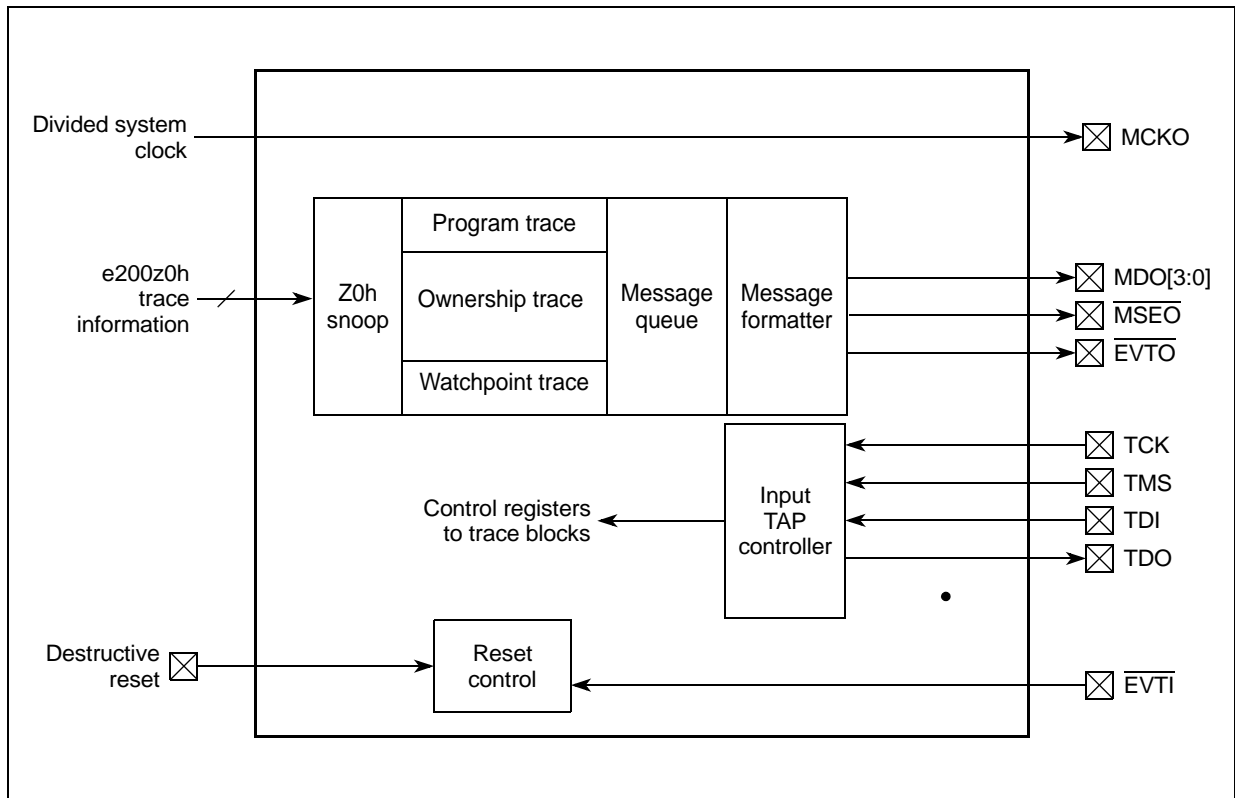


Figure 574. NDI functional block diagram

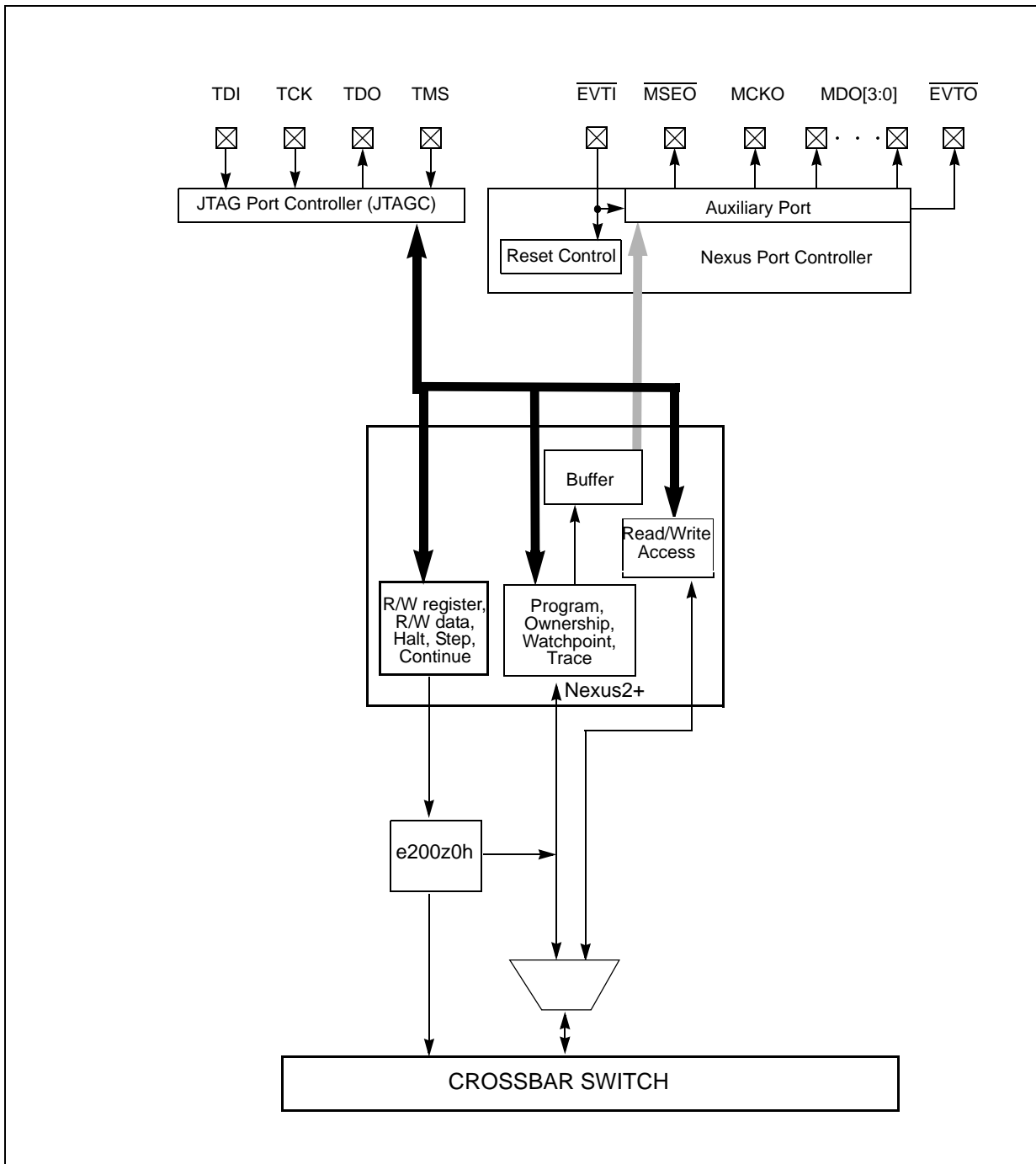


Figure 575. NDI implementation block diagram

### 36.2.1 NDI features

The NDI module of the SPC560B54/6x is compliant with the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- 12 pin interface for high throughput, including existing four JTAG pins
  - Two modes are supported: full port mode (FPM) and reduced port mode (RPM). FPM comprises 4 MDO pins. RPM comprises 2 MDO pins.
  - Auxiliary output port
    - One MCKO (message clock out) pin
    - 4 MDO (message data out) pins
    - One  $\overline{\text{MSEO}}$  (message start/end out) pins
    - One EVTO (event out) pin
  - Auxiliary input port uses one  $\overline{\text{EVTI}}$  (event in) pin
  - JTAG port uses four pins (TDI, TDO, TMS, and TCK)
- The NPC block performs the following functions:
  - Nexus Device Identification Register and Messaging
  - Generates MCKO enable and frequency division control signals
  - Generates an MCKO clock gating control signal to enable gating of MCKO when the auxiliary output port is idle.
  - Control of the device-wide debug mode
  - Generates asynchronous reset signal for Nexus blocks
- Host processor (e200z0h) development support features
  - IEEE-ISTO 5001-2003 standard class 2+ compliant.
    - Program trace via [Branch Trace Messaging \(BTM\)](#), with the option of using Branch history messaging to enhance message throughput.
    - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing development tools to trace ownership flow.
    - Run-time access to the on-chip memory map via the JTAG port.
    - Watchpoint messaging (WPM) via the auxiliary port. This allows a watchpoint to be set, which then sends a watchpoint message each time the watchpoint is hit. Unlike watchpoint triggering, WPM does not stop the core or start trace.
    - Registers for Program Trace, Ownership Trace and Watchpoint Trigger.
  - Additional class 4 features
    - Watchpoint trigger enable of program and/or data trace messaging. This is an extension of WPM to allow a watchpoint to stop or start program or data trace.
    - Processor overrun control
  - All features controllable and configurable via JTAG port.

### 36.2.2 Modes of operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the destructive reset signal, or through state machine transitions controlled by TMS. Ownership of the TAP

is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAG controller (JTAGC) block.

The NPC transitions out of the reset state immediately following negation of power-on reset.

### 36.2.2.1 Nexus reset mode

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The auxiliary output port pins are negated if the NDI controls the pads.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.
- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to three-state the output pins or use them for another function.

### 36.2.2.2 Full-port mode

In the full-port mode (FPM), 4 MDO pins (MDO[0...3]) pins are used to transmit Nexus messages. All trace features are available by writing to the configuration registers via the JTAG port. FPM is entered by asserting the MCKO\_EN and FPM bits in the [Port Configuration Register \(PCR\)](#).

### 36.2.2.3 Reduced-port mode

In the reduced-port mode (RPM), the number of MDO pins is reduced from 4 to 2 (MDO[0...1]). All trace features are available by writing to the configuration registers via jtag. Trace buffer overflow is more likely in the reduced port mode.

### 36.2.2.4 Disabled-port mode

In the disabled-port mode, auxiliary output pin port enable signals are negated, thereby disabling the message transmission. Any debug feature that generates messages cannot be used. The primary features available are class 1 features and read/write access to the registers. Class 1 features include the ability to trigger a breakpoint event indication through EVTO.

## 36.3 External signal description

The auxiliary and JTAG pin interfaces provide for the transmission of messages from Nexus modules to the external development tools and for access to Nexus client registers. The auxiliary/JTAG pin definitions are outlined in [Table 542](#).

**Table 542. Signal properties**

Name	Port	Function
EVTO	Auxiliary	Event Out pin
EVTI	Auxiliary	Event In pin
MCKO	Auxiliary	Message Clock Out pin (from NPC)
MDO[3:0]	Auxiliary	Message Data Out pins

**Table 542. Signal properties (continued)**

Name	Port	Function
MSEO	Auxiliary	Message Start/End Out pins
TCK	JTAG	Test Clock Input
TDI	JTAG	Test Data Input
TDO	JTAG	Test Data Output
TMS	JTAG	Test Mode Select Input

See the signal description chapter of this reference manual for detailed signal descriptions.

### 36.4 Memory map and registers

The NDI block contains no memory mapped registers. Nexus registers are accessed by the development tool via the JTAG port using a client select and a register index. The client select is controlled by loading the correct access instruction into the JTAG controller. After the desired client TAP is selected, OnCE registers for that client are accessible by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD). Nexus is enabled, and the associated Nexus registers become accessible, by loading the NEXUS\_ENABLE instruction into the RS[0:6] field of the OCMD. When Nexus register access is enabled, the desired Nexus register is accessible using the index shown in [Table 543](#).

**Table 543. Nexus Client JTAG Instructions**

Instruction	Description	Opcode
<b>NPC JTAG Instruction Opcodes</b>		
NEXUS_ENABLE	Opcode for NPC Nexus Enable instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
<b>e200z0h OnCE JTAG Instruction Opcodes</b>		
NEXUS_ACCESS	Opcode for e200z0h OnCE Nexus Enable instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z0h OnCE BYPASS instruction (10-bits)	0x7F

**Table 544. NDI Registers**

Nexus Register	Nexus Access Opcode	Read/Write	Read Address	Write Address
<b>NPC Registers</b>				
Device ID (DID)		R	0x00	0x05
Port Configuration Register (PCR)		R/W	0x7F	0x07F
<b>e200z0h Control/Status Registers</b>				
Development Control 1 (DC1)	0x2	R/W	0x04	0x05
Development Control 2 (DC2)	0x3	R/W	0x06	0x07
Development Status (DS)	0x4	R	0x08	—
Read/Write Access Control/Status (RWCS)	0x7	R/W	0x0E	0x0F
Read/Write Access Address (RWA)	0x9	R/W	0x12	0x13
Read/Write Access Data (RWD)	0xA	R/W	0x14	0x15
Watchpoint Trigger (WT)	0xB	R/W	0x16	0x17
Reserved	0xC-> 0x3F	R/W	0x18-> 0x7E	0x19-> 0x7F

### 36.5 NDI functional description

The NDI block is implemented by integrating the following blocks:

- Nexus Port Controller Block
- Nexus e200z0h Development Interface (OnCE and Nexus2+ subblocks)
- NPC\_HNDSHK module

*Note:* The TAP controller logic, reset logic, and some miscellaneous logic are duplicated in all these blocks.

#### 36.5.1 NPC\_HNDSHK module

This module enables debug entry/exit across low power modes (Stop, Halt, standby).

The NPC\_HNDSHK supports:

- Setting and clearing of the NPC PCR sync bit on low-power mode entry and exit
- Putting the core into debug mode on low-power mode exit
- Generating a falling edge on the JTAG TDO pad on low-power mode exit

On HALT, STOP, or STANDBY mode entry, the MC\_ME asserts the lp\_mode\_entry\_req input after the clock disable process has completed and before the processor enters its halted or stopped state. The mode transition will then not proceed until the lp\_mode\_entry\_ack output has been asserted. The notification to the debugger of a low-power mode entry consists of setting the low-power mode handshake bit in the port control register (read by the debugger) via the lp\_sync\_in output. The debugger acknowledges that the transition into a low-power mode may proceed by clearing the low-power mode



handshake bit in the port control register (written by the debugger), which results in the deassertion of the `lp_sync_out` input.

In anticipation of the low-power mode exit notification, the TDO pad is driven to `1'.

On HALT or STOP mode exit, the MC\_ME asserts the `lp_mode_exit_req` input after ensuring that the regulator and memories are in normal mode and before the processor exits its halted or stopped state. The mode transition will then not proceed until the `lp_mode_exit_ack` output has been asserted. The MC\_RGM asserts the `exit_from_standby` input when executing a reset sequence due to a STANDBY exit. The reset sequence will then not complete until the `lp_mode_exit_ack` output has been asserted.

The notification to the debugger of a low-power mode exit consists of driving the TDO pad to `0'. The debugger acknowledges that the transition from a low-power mode can continue by setting the low-power mode sync bit in the port control register (written by debugger), which results in the assertion of the `lp_sync_out` input.

*Note:* *The debugger clock multiplexer may not guarantee glitch free switching. Therefore, TCK should be disabled from when the debugger clears the sync bit in ENTRY\_CLR until the debugger senses the falling edge of TDO in TDO\_SET.*

### 36.5.2 Enabling Nexus Clients for TAP Access

Once the NDI is out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. Once the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

### 36.5.3 TAP Sharing

Each of the individual Nexus blocks on the MCU implements a TAP controller for accessing its registers. The JTAGC controls the ownership of the TAP so that the interface to all of these individual TAP controllers appears to be a single port from outside the device. Once a Nexus client has been granted ownership of the TAP, any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

### 36.5.4 Configuring the NDI for Nexus Messaging

The NDI is placed in disabled mode upon exit of power-on reset. If message transmission via the auxiliary port is desired, a write to the port configuration register (PCR) located in the NPC is then required to enable the NDI and select the mode of operation. Asserting MCKO\_EN in the PCR places the NDI in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO\_DIV field. Asserting the FPM bit selects full-port mode.

When writing to the PCR, the PCR LSB must be written to a logic 0. Setting the LSB of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

[Table 545](#) describes the NDI configuration options.

**Table 545. NDI Configuration Options**

MCKO_EN bit of PCR	FPM bit of PCR	Configuration
0	X	Disabled
1	1	Full-Port Mode
1	0	Reduced Port Mode

### 36.5.5 Programmable MCKO Frequency

MCKO is an output clock to the development tools used for the timing of  $\overline{MSEO}$  and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO\_DIV field in the port configuration register (PCR) located in the NPC. Possible operating frequencies include SYS\_CLK, one-half, one-quarter, and one-eighth SYS\_CLK speed.

*Table 546* shows the MCKO\_DIV encodings. In this table, SYS\_CLK represents the system clock frequency.

**Table 546. MCKO\_DIV Values**

MCKO_DIV[2:0]	MCKO Frequency
0b000 <sup>(1)</sup>	SYS_CLK
0b001	SYS_CLK/2
0b010	Reserved
0b011	SYS_CLK/4
0b100	Reserved
0b101	Reserved
0b110	Reserved
0b111	SYS_CLK/8

1. The SYS\_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.

*Note:* The Nexus Port Controller Port Configuration Register MCKO Divider bits (NPC\_PCR[MCKO\_DIV]) can be set to 0b000 to select a 1X clock rate as the Nexus Auxiliary output port frequency for the MCKO and MDO pins. Depending on the system frequency, this may force the MCKO and MDO pins to switch at a frequency higher than can be supported by the pins. This maximum frequency is specified in the device electrical specification of the Nexus MCKO and MDO pins. Insure that the maximum operating frequency of the MDO and MCKO pins is not violated when setting the NPC\_PCR[MCKO\_DIV] values.

## 36.6 Nexus Port Controller (NPC)

The Nexus Port Controller (NPC) is that part of the NDI that controls access and arbitration of the device’s internal Nexus modules. The NPC contains the port configuration register (PCR) and the device identification register (DID).



### 36.6.1 NPC Features

The NPC performs the following functions:

- Controls arbitration for ownership of the Nexus auxiliary output port in case of multiple nexus clients. SPC560B54/6x has only single Nexus 2+ client implemented.
- Nexus device identification register and messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of EVTO in case of multiple Nexus clients

### 36.6.2 NPC Memory Map

[Table 547](#) shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC does not implement the client select control register because the value does not matter when accessing the registers. Note that the bypass register (refer to [Section 36.6.3.1: Bypass Register](#)) and instruction register (refer to [Section 36.6.3.2: Instruction Register](#)) have no index values. These registers are not accessed in the same manner as Nexus client registers.

**Table 547. NPC Memory Map**

Index	Register Name	Register Description	Size (bits)
0	DID	Device ID register	32
127	PCR	Port configuration register	32

### 36.6.3 NPC Register Descriptions

This section consists of NPC register descriptions.

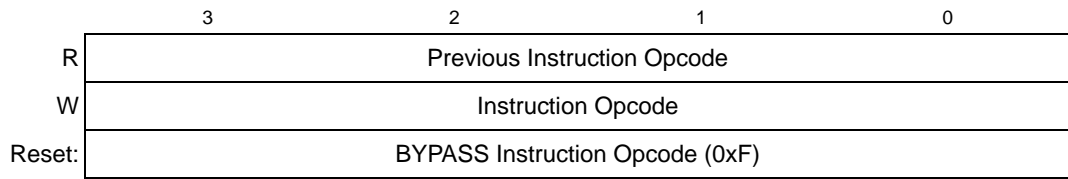
#### 36.6.3.1 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the CAPTURE-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

#### 36.6.3.2 Instruction Register

The NPC uses a 4-bit instruction register as shown in [Figure 576](#). The instruction register is accessed via the SELECT\_IR\_SCAN path of the tap controller state machine, and allows instructions to be loaded into the module to enable the NPC for register access (NEXUS\_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

As shown in [Section 35.8.3: TAP controller state machine](#), instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.



**Figure 576. 4-Bit Instruction Register**

**36.6.3.3 Nexus Device ID Register (DID)**

The NPC device identification register allows the part revision number, design center, part identification number, and manufacturer identity code of the device to be determined through the auxiliary output port, and serially through TDO. See [Section 36.6.4.3.1: NPC IEEE 1149.1-2001 \(JTAG\) TAP](#). This register is read-only.

**Table 548. DID Field Descriptions**

Field	Description
PRN	Part Revision Number. Contains the revision number of the part. This field changes with each revision of the device or module.
DC	Design Center. Indicates the Freescale design center.
PIN	Part Identification Number. The value is 0x248.
MIC	Manufacturer Identity Code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID.
bit 0	Fixed Per JTAG 1149.1. Always set to 1.

**36.6.3.4 Port Configuration Register (PCR)**

The PCR is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register should be configured as soon as the NDI is enabled.

The PCR register may be rewritten by the debug tool subsequent to the enabling of the NPC for low power debug support. In this case, the debug tool may set and clear the LP\_DBG\_EN and SLEEP\_SYNC bits, but must preserve the original state of the remaining bits in the register.

*Note: The mode (MCKO\_GT) or clock division (MCKO\_DIV) bits must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.*

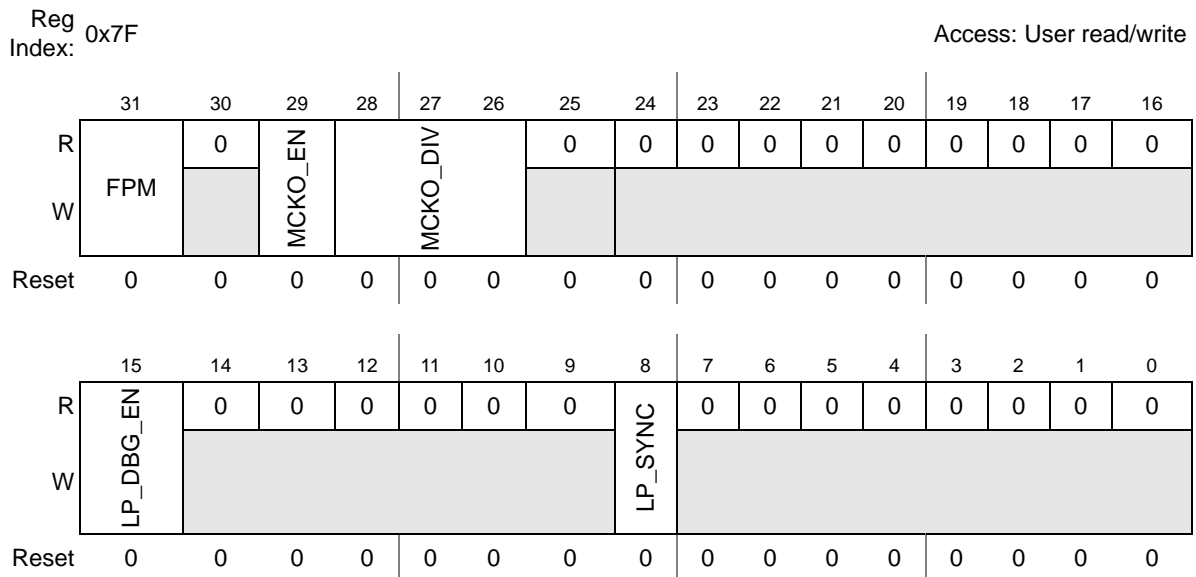


Figure 577. Port Configuration Register (PCR)

Table 549. PCR Field Descriptions

Field	Description
FPM	Full Port Mode. The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 = 2 MDO pins are used to transmit messages. 1 = 4 MDO pins are used to transmit messages. <b>Note:</b> This bit must always be set when configuring the NDI.
MCKO_EN	MCKO Enable. This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.
MCKO_DIV	MCKO Division Factor. The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. In this table, SYS_CLK represents the system clock frequency. Refer to <a href="#">Table 546</a> for details.

**Table 549. PCR Field Descriptions (continued)**

Field	Description
LP_DBG_EN <sup>(1)</sup>	Low Power Debug Enable. The LP_DBG_EN bit enables debug functionality to support entry and exit from the Standby mode. 0 Low power debug disabled. 1 Low power debug enabled.
LP_SYNC <sup>(1)</sup>	Low power mode Synchronization. The LP_SYNC bit is used to synchronize the entry into / exit from low power mode (HALT/STOP/STANDBY) between the device and debug tool. Before entry into low power mode, the device sets this bit. After reading LP_SYNC as set, the debug tool then clears LP_SYNC to acknowledge to the device that it may enter into low power mode. During exit from low power mode, the debug tool sets LP_SYNC to acknowledge to the device that it may exit low power mode. 0 No low power mode entry pending. 1 low power mode entry pending. <b>Note:</b> During low power entry, the device sets the LP_SYNC bit. The debug tool then clears this bit to enter low power mode. The bit changes to 0, but is not readable, since the device immediately enters low power mode. During low power exit, the device then clears the LP_SYNC bit. Then the debug tool sets this bit to exit low power mode. The bit does not change to 1.

1. The JTAG Clock (TCK) typically operates at a frequency well below the system clock frequency, as specified in the electrical data sheets. In some cases, however, such as low power mode (if the device supports low power modes), the system clock frequency may be lowered significantly from the normal operating range. If the system clock frequency is reduced below the frequency of TCK it will no longer be possible to communicate with the Nexus Port Controller Port Configuration Register (NPC\_PCR). Therefore, if the tool needs to update the NPC\_PCR Low Power Debug Enable (NPC[PCR[LP\_DBG\_EN]) or Low Power Synchronization bits (NPC[PCR[LP\_SYNC]], the TCK clock frequency must be lowered.

### 36.6.4 NPC Functional Description

#### 36.6.4.1 NPC Reset Configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO\_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO\_DIV field.

[Table 545](#) describes the NPC reset configuration options.

#### 36.6.4.2 Auxiliary Output Port

In case of multiple Nexus clients, the auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the individual modules and arbitrates for access to the port.

##### 36.6.4.2.1 Output Message Protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the MSEO functions. The MSEO pin is used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and MSEO are sampled on the rising edge of MCKO.

[Figure 578](#) illustrates the state diagram for MSEO transfers. All transitions not included in the figure are reserved, and must not be used.

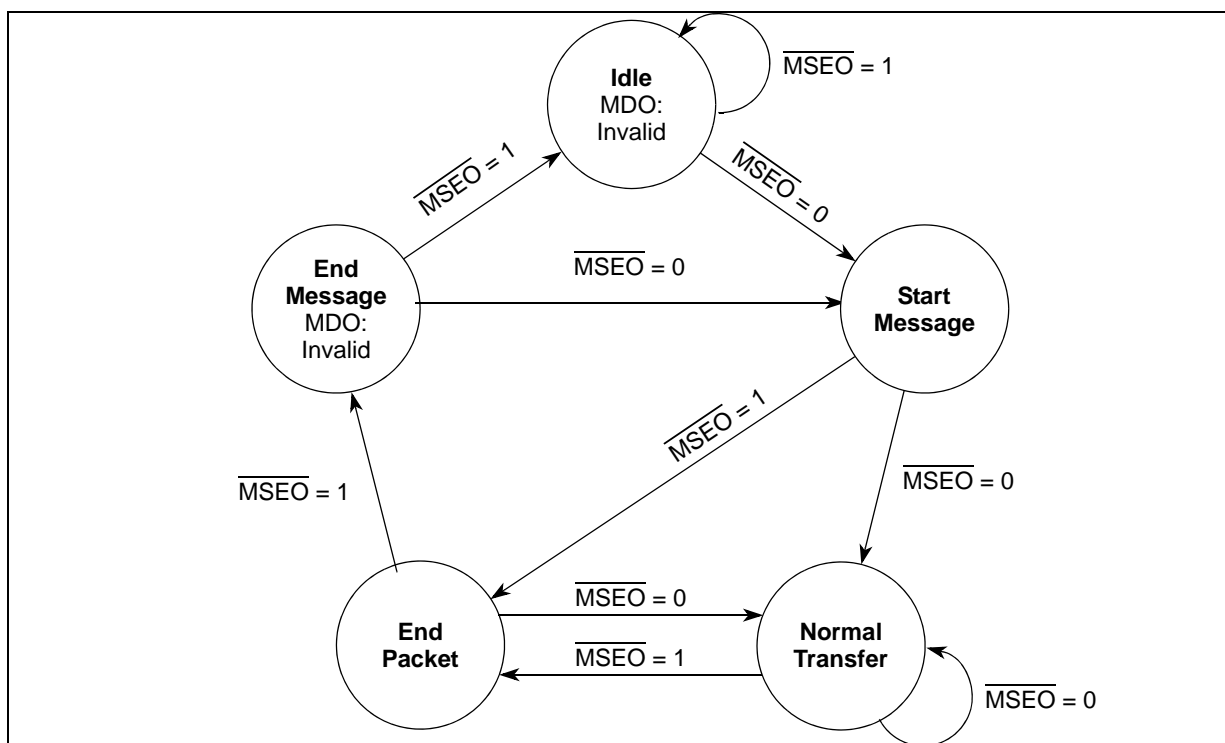


Figure 578. MSEO Transfers

36.6.4.2.2 Output Messages

36.6.4.3 Output Messages

In addition to sending out messages generated in other Nexus blocks, the NPC can also output the device ID message contained in the device ID register and the port replacement output message on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 550 describes the device ID and port replacement output messages that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 550. NPC Output Messages

Message Name	Min. Packet Size (bits)	Max Packet Size (bits)	Packet Type	Packet Name	Packet Description
Device ID Message	6	6	fixed	TCODE	Value = 1
	32	32	fixed	ID	DID register contents

Figure 579 shows the various message formats that the pin interface formatter has to encounter. Note that for variable-length fields, the transmitted size of the field is determined from the range of the least significant bit to the most significant non-zero-valued bit (i.e. most significant zero-valued bits are not transmitted).

Figure 579. Message Field Sizes

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. Size <sup>1</sup> (bits)	Max Size <sup>2</sup> (bits)
Device ID Message	1	Fixed = 32	NA	NA	NA	NA	38	38

36.6.4.3.0.1 Rules of Messages

The rules of messages include the following:

- A variable-sized field within a message must end on a port boundary. This implies, all messages are formatted by the transmitting client to align on the port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field may start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets may start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.
- Within a field, the lowest significant bits are shifted out first. [Figure 580](#) shows the transmission sequence of a message that is made up of a TCODE followed by three fields.

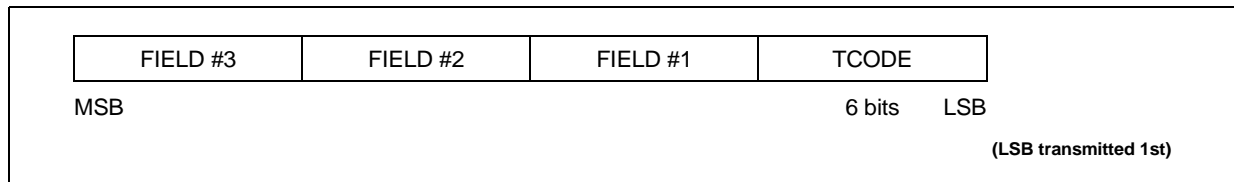


Figure 580. Transmission Sequence of Messages

36.6.4.3.1 NPC IEEE 1149.1-2001 (JTAG) TAP

The NPC uses the IEEE 1149.1-2001 TAP, which uses the state machine for accessing registers. The NPC also implements the Nexus controller state machine as defined by the IEEE-ISTO 5001-2003 standard as shown in [Figure 581](#).

The instructions implemented by the NPC TAP controller are listed in [Table 543](#).

36.6.4.3.1.1 Enabling the NPC TAP Controller

Assertion of the destructive reset resets the NPC TAP controller. When not in reset phase, the NPC TAP controller is enabled by loading the ACCESS\_AUX\_TAP\_NPC instruction in the JTAGC. Loading the NEXUS-ENABLE instruction then grants access to NPC registers.



### 36.6.4.3.1.2 Loading NEXUS-ENABLE Instruction

Access to the NPC registers is enabled by loading the NPC NEXUS-ENABLE instruction when NPC has ownership of the TAP. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in [Figure 581](#), transitions to the REG\_SELECT state. The Nexus controller has three states: idle, register select, and data access. [Table 551](#) illustrates the IEEE® 1149.1 sequence to load the NEXUS-ENABLE instruction.

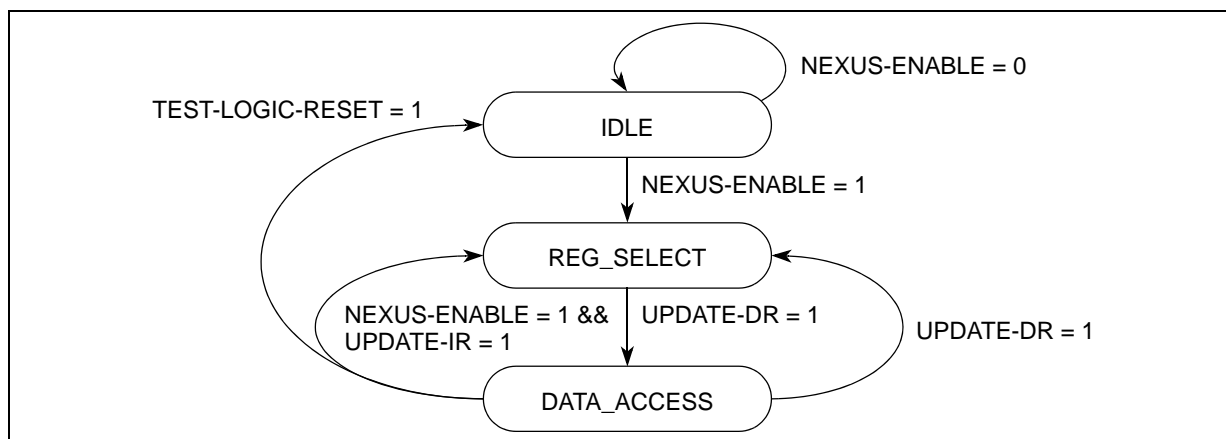


Figure 581. NEXUS Controller State Machine

Table 551. Loading NEXUS-ENABLE Instruction

Clock	TDI	TMS	IEEE 1149.1 State	Nexus State	Description
0	—	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	—	1	SELECT-DR-SCAN	IDLE	Transitional state
2	—	1	SELECT-IR-SCAN	IDLE	Transitional state
3	—	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	—	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE® 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
3 TCKs in SHIFT-IR					
8	0	1	EXIT1-IR	IDLE	Last bit of instruction shifted in
9	—	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
10	—	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

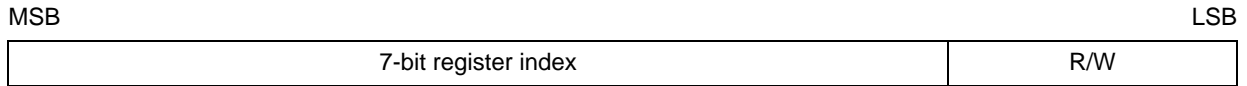
### 36.6.4.3.1.3 Selecting a Nexus Client Register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path of the IEEE® 1149.1–2001 TAP controller state machine. The Nexus controller defaults to the REG\_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN

path: one pass to select the register and the second pass to read/write the register. (See [Section 35.8.3: TAP controller state machine](#).)

The first pass through the SELECT-DR-SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the LSB followed by a 7-bit register address index, as illustrated in [Figure 582](#). The read/write control bit is set to 1 for writes and 0 for reads.



**Figure 582. IEEE, 1149.1 Controller Command Input**

The second pass through the SELECT-DR-SCAN path is used to read or write the register data by shifting in the data (LSB first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE® 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE® 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, no requirement exists to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

[Table 552](#) illustrates a sequence that writes a 32-bit value to a register.

**Table 552. Write to a 32-Bit Nexus Client Register**

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
12	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
13	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
14	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path
15	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
16	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
48	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.
49	1	UPDATE-DR	DATA_ACCESS	Value written to register
50	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

### 36.6.4.3.2 Nexus JTAG Port Sharing

The Nexus module on the device implements a TAP controller for accessing its registers. Only the module whose ACCESS\_AUX\_TAP instruction is loaded has control of the TAP. This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. Once a Nexus module has ownership of the TAP, that module acts like a single-bit shift register, or bypass register, if no register is selected as the shift path.

### 36.6.4.3.3 MCKO

MCKO is an output clock to the development tools used for the timing of  $\overline{\text{MSE0}}$  and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO\_DIV[2:0] field in the PCR. Possible operating frequencies include full, one-half, one-quarter, and one-eighth SYS\_CLK speed. MCKO is enabled by setting the MCKO\_EN bit in the PCR.

### 36.6.4.3.4 Nexus Reset Control

The destructive reset is used as the primary reset signal for the NPC.

## 36.6.5 NPC Initialization/Application Information

To initialize the TAP for NPC register accesses, the following sequence is required:

1. Enable the NPC TAP controller. This is achieved by loading the ACCESS\_AUX\_TAP\_NPC instruction in the JTAGC.
2. Load the TAP controller with the NEXUS-ENABLE instruction.

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE®-ISTO 5001-2003 standard for more detail.

## 36.7 Nexus Module (Nexus2+)

The Nexus2+ module provides real-time development capabilities for the device core in compliance with the IEEE-ISTO Nexus 5001-2003 standard. This module provides development support capabilities without requiring the use of address and data pins for internal visibility.

A portion of the pin interface (the JTAG port) is also shared with the OnCE / Nexus 1 unit. The IEEE-ISTO 5001-2003 standard defines an extensible auxiliary port that is used in conjunction with the JTAG port in e200z0h processor.

### 36.7.1 Nexus2+ Introduction

This section defines the auxiliary pin functions, transfer protocols and standard development features of the Nexus2+ module. The development features supported are Program trace, watchpoint messaging, ownership trace, and read/write access via the JTAG interface. The Nexus2+ module implements one class3 feature (run time date access) and two class 4 features: Watchpoint Triggering and Processor Overrun Control.

### 36.7.2 Nexus2+ Block Diagram

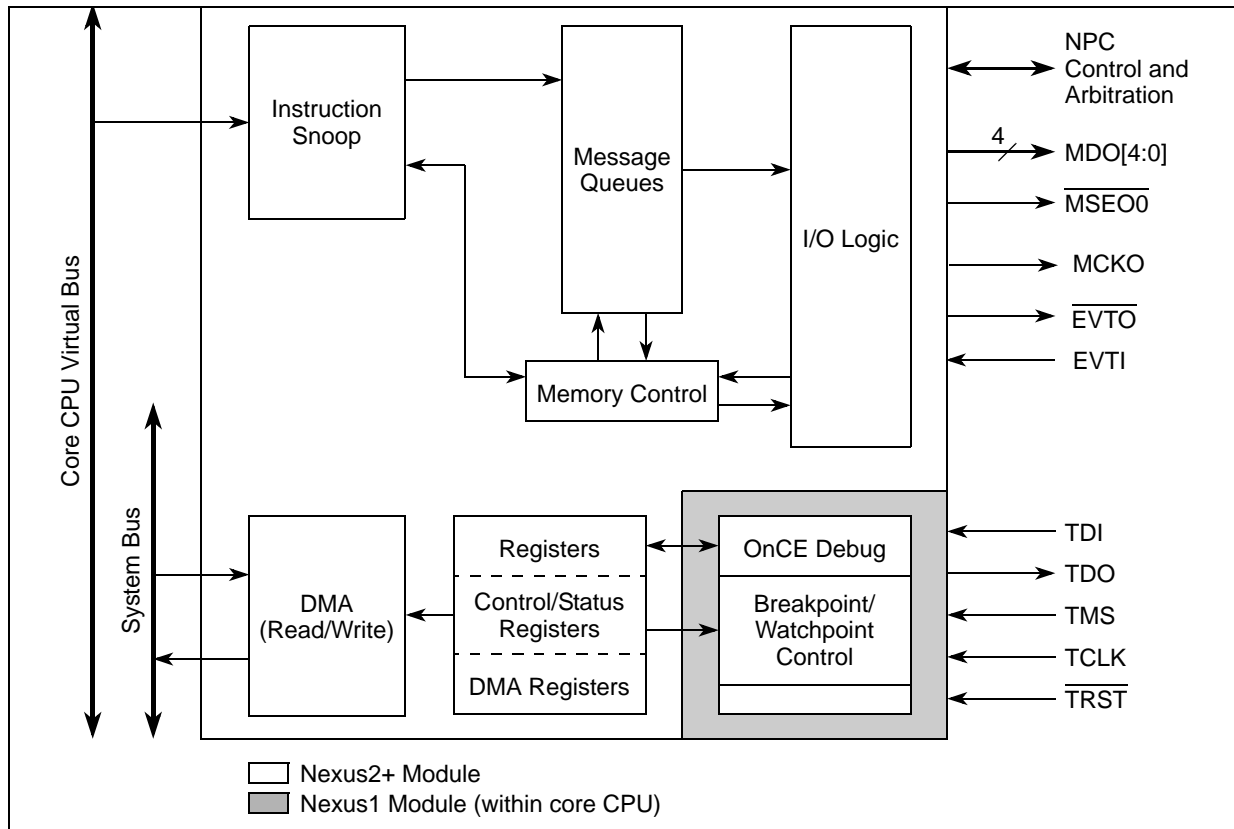


Figure 583. e200z0 Nexus2+ Functional Block Diagram

### 36.7.3 Nexus2+ Features

The Nexus2+ module is compliant with Class 2 of the IEEE-ISTO 5001-2003 standard, with additional Class 3 and 4 features available. The following features are implemented:

- Program trace via [Branch Trace Messaging \(BTM\)](#), with the option of using Branch history messaging to enhance message throughput.
- Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated.

An ownership trace message is transmitted when a new process/task is activated, allowing development tools to trace ownership flow.

- Run-time access to the on-chip memory map via the JTAG port.
- Watchpoint messaging (WPM) via the auxiliary port. This allows a watchpoint to be set, which then sends a watchpoint message each time the watchpoint is hit. Unlike watchpoint triggering, WPM does not stop the core or start trace.
- Registers for Program Trace, Ownership Trace and Watchpoint Trigger.
- Additional class 4 features
  - Watchpoint trigger enable of program and/or data trace messaging. This is an extension of WPM to allow a watchpoint to stop or start program or data trace.
  - Processor overrun control
- All features controllable and configurable via JTAG port.

### 36.7.4 Enabling Nexus2+ Operation

The Nexus module is enabled by loading a single instruction (ACCESS\_AUX\_TAP\_ONCE) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS2\_ACCESS instruction. For the e200z0h Class 2+ Nexus module, the OCMD value is 0b00\_0111\_1100. Once enabled, the module is ready to accept control input via the JTAG pins.

Enabling the Nexus 2+ module automatically enables the generation of Debug Status Messages.

The Nexus module is disabled when the JTAG state machine reaches the test-logic-reset state. This state can be reached by cycling through the state machine using the TMS pin. The Nexus module also is disabled if destructive reset occurs. If the Nexus2+ module is disabled, no trace output is provided, and the module disables (drives inactive) auxiliary port output pins MDO[3:0], MSEO, MCKO. Nexus registers are not available for reads or writes.

### 36.7.5 TCODEs Supported by Nexus2+

The Nexus2+ pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001-2003 standard defines a set of public messages. The Nexus2+ module supports the public TCODEs seen in [Table 553](#). Each message contains multiple packets transmitted in the order shown in the table.

**Table 553. Public TCODEs Supported by Nexus2+**

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Debug Status	6	6	TCODE	Fixed	TCODE number = 0 (0x00)
	4	4	SRC	Fixed	source processor identifier
	8	8	STATUS	Fixed	Debug status register (DS[31:24])

**Table 553. Public TCODEs Supported by Nexus2+ (continued)**

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Ownership Trace Message	6	6	TCODE	Fixed	TCODE number = 2 (0x02)
	4	4	SRC	Fixed	source processor identifier
	32	32	PROCES S	Fixed	Task/Process ID tag
Program Trace — Direct Branch Message <sup>(1)</sup>	6	6	TCODE	Fixed	TCODE number = 3 (0x03)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variabl e	# sequential instructions executed since last taken branch
Program Trace — Indirect Branch Message	6	6	TCODE	Fixed	TCODE number = 4 (0x04)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variabl e	# sequential instructions executed since last taken branch
	1	32	U-ADDR	Variabl e	unique part of target address for taken branches/exceptions
Error Message	6	6	TCODE	Fixed	TCODE number = 8 (0x08)
	4	4	SRC	Fixed	source processor identifier
	5	5	ECODE	Fixed	error code
Program Trace — Direct Branch Message w/ Sync <sup>(1)</sup>	6	6	TCODE	Fixed	TCODE number = 11 (0x0B)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variabl e	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variabl e	full target address (leading zeros truncated)
Program Trace — Indirect Branch Message w/ Sync <sup>(1)</sup>	6	6	TCODE	Fixed	TCODE number = 12 (0x0C)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variabl e	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variabl e	full target address (leading zeros truncated)
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0x0F)
	4	4	SRC	Fixed	source processor identifier
	8	8	WPHIT	Fixed	# indicating watchpoint sources

**Table 553. Public TCODEs Supported by Nexus2+ (continued)**

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Resource Full Message	6	6	TCODE	Fixed	TCODE number = 27 (0x1B)
	4	4	SRC	Fixed	source processor identifier
	4	4	RCODE	Fixed	resource code (Refer to RCODE values in <a href="#">Table 555</a> ) - indicates which resource is the cause of this message
	1	32	RDATA	Variable	branch / predicate instruction history
Program Trace — Indirect Branch History Message	6	6	TCODE	Fixed	TCODE number = 28 (0x1C) (see footnote 1 below)
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	unique part of target address for taken branches/exceptions
	1	32	HIST	Variable	branch / predicate instruction history
Program Trace — Indirect Branch History Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 29 (0x1D) <sup>(1)</sup>
	4	4	SRC	Fixed	source processor identifier
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	full target address (leading zero (0) truncated)
	1	32	HIST	Variable	branch / predicate instruction history
Program Trace — Program Correlation Message	6	6	TCODE	Fixed	TCODE number = 33 (0x21)
	4	4	SRC	Fixed	source processor identifier
	4	4	EVCODE	Fixed	event correlated w/ program flow (Refer to <a href="#">Table 556</a> )
	1	8	I-CNT	Variable	# sequential instructions executed since last taken branch
	1	32	HIST	Variable	branch / predicate instruction history

1. The user can select between the two types of program trace. If the branch history method is selected, the shaded TCODEs above are not messaged out.

[Table 554](#) shows the error code encodings used when reporting an error via the Nexus2+ Error Message.

**Table 554. Error Code Encoding (TCODE = 8)**

Error Code (ECODE)	Description
00000	Ownership trace overrun
00001	Program trace overrun
00010	Reserved
00011	Read/write access error
00101	Invalid access opcode (Nexus register unimplemented)
00110	Watchpoint overrun
00111	Program trace and ownership trace overrun
01000	(Program trace or ownership trace) and watchpoint overrun
01001–01111	Reserved
11000	BTM lost due to collision w/ higher priority message
11001–11111	Reserved

Table 555 shows the encodings used for resource codes for certain messages.

**Table 555. RCODE values (TCODE = 27)**

Resource Code (RCODE)	Description	Resource Data (RDATA)
0000	Program Trace Instruction Counter overflow (reached 255 and was reset)	0xFF
0001	Program Trace, Branch / Predicate Instruction History. This type of packet is terminated by a stop bit set to 1 after the last history bit.	Branch History. This type of packet is terminated by a stop bit set to a 1 after the last history bit.

Table 556 shows the event code encodings used for certain messages.

**Table 556. Event Code Encoding (TCODE = 33)**

Event Code	Description
0000	Entry into Debug Mode
0001	Entry into Low Power Mode (CPU only) <sup>(1)</sup>
0010–0011	Reserved for future functionality
0100	Disabling Program Trace
0101–1111	Reserved for future functionality

1. The device enters Low Power Mode when the Nexus stall mode is enabled (NEXUS2\_DC1[OVC] = 0b011) and a trace message is in danger of over-flowing the Nexus queue.



### 36.7.6 Nexus2+ Memory Map

This section describes the Nexus2+ programmer’s model. Nexus2+ registers are accessed using the JTAG/OnCE port in compliance with IEEE 1149.1. See [Section 36.7.8: Nexus2+ Register Access via JTAG / OnCE](#) for details on Nexus2+ register access.

*Note:* Nexus2+ registers and output signals are numbered using bit 0 as the least significant bit. This bit ordering is consistent with the ordering defined by the IEEE-ISTO 5001 standard.

[Table 557](#) details the register map for the Nexus2+ module.

**Table 557. Nexus2+ Memory Map**

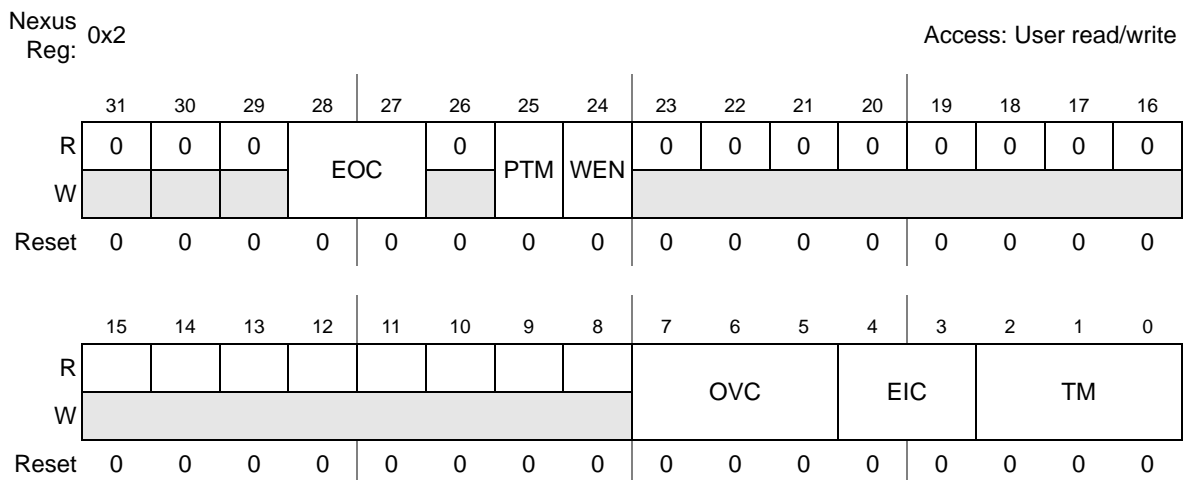
Index	Register Description	Read Address <sup>(1)</sup>	Write Address <sup>1</sup>
0x02	e200z0h Development Control1 (PPC_DC1)	0x04	0x05
0x03	e200z0h Development Control2 (PPC_DC2)	0x06	0x07
0x04	e200z0h Development Status (PPC_DS)	0x08	—
0x07	e200z0h Read/Write Access Control/Status (Nexus2_RWCS)	0x0E	0x0F
0x09	e200z0h Read/Write Access Address (Nexus2_RWA)	0x12	0x13
0x0A	e200z0h Read/Write Access Data (Nexus2_RWD)	0x14	0x15
0x0B	e200z0h Watchpoint Trigger (PPC_WT)	0x16	0x17
0xC – 0x3F	Reserved		

1. See [Section 36.6.4.3.1: NPC IEEE 1149.1-2001 \(JTAG\) TAP](#), for a description of the read and write address usage for the e200z0h Nexus Control/Status registers.

### 36.7.7 Nexus2+ Register Definition

#### 36.7.7.1 Development Control Register 1, 2 (DC1, DC2)

The development control registers are used to control the basic development features of the Nexus2+ module. Development control register 1 is shown in [Figure 584](#) and its fields are described in [Table 558](#).

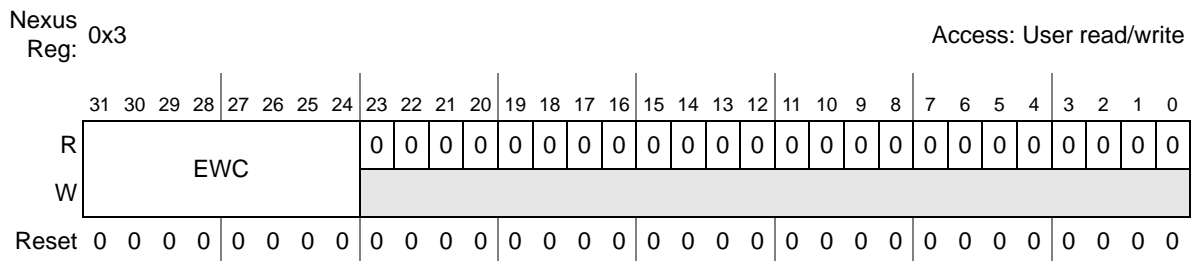


**Figure 584. Development Control Register 1 (DC1)**

**Table 558. DC1 Field Descriptions**

Field	Description
EOC[1:0]	$\overline{\text{EVT0}}$ control. 00 $\overline{\text{EVT0}}$ upon occurrence of watchpoints (configured in DC2) 01 $\overline{\text{EVT0}}$ upon entry into debug mode 10 $\overline{\text{EVT0}}$ upon timestamping event 11 Reserved
PTM	Program trace method. 0 Program trace uses traditional branch messages 1 Program trace uses branch history messages
WEN	Watchpoint trace enable. 0 Watchpoint Messaging disabled 1 Watchpoint Messaging enabled
OVC[2:0]	Overrun control. 000 Generate overrun messages 001–010 Reserved 011 Delay processor for BTM / OTM overruns 1XX Reserved
EIC[1:0]	$\overline{\text{EVTI}}$ control. 00 $\overline{\text{EVTI}}$ is used for synchronization (program trace/ data trace) 01 $\overline{\text{EVTI}}$ is used for debug request 1X Reserved
TM[2:0]	Trace mode. Any or all of the TM bits may set, enabling one or more traces. 000 No trace 1XX Program trace enabled X1X Reserved XX1 Ownership trace enabled

Development control register 2 is shown in [Figure 585](#) and its fields are described in [Table 559](#).



**Figure 585. Development Control Register 2 (DC2)**

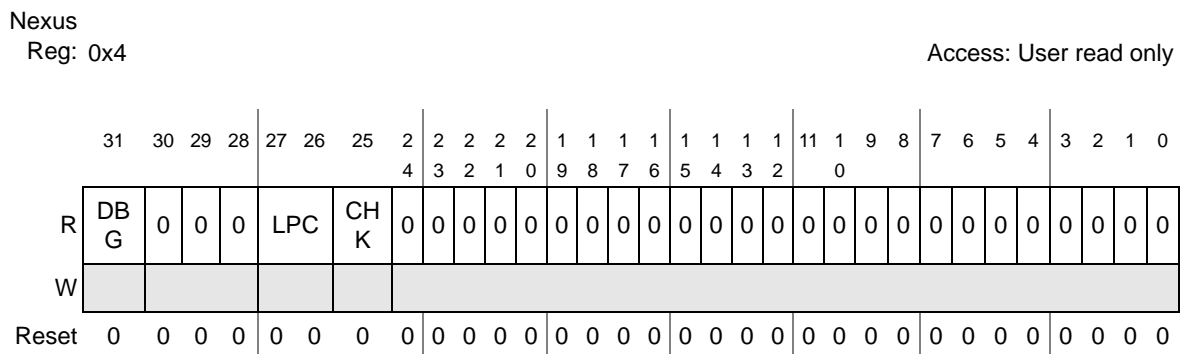
**Table 559. DC2 Field Descriptions**

Field	Description
EWC[7:0]	<p><math>\overline{EVTO}</math> watchpoint configuration. Any or all of the bits in EWC may be set to configure the <math>\overline{EVTO}</math> watchpoint.</p> <p>00000000 No Watchpoints trigger <math>\overline{EVTO}</math></p> <p>1XXXXXXXX Watchpoint #0 (IAC1 from Nexus1) triggers <math>\overline{EVTO}</math></p> <p>X1XXXXXXXX Watchpoint #1 (IAC2 from Nexus1) triggers <math>\overline{EVTO}</math></p> <p>XX1XXXXXX Watchpoint #2 (IAC3 from Nexus1) triggers <math>\overline{EVTO}</math></p> <p>XXX1XXXXX Watchpoint #3 (IAC4 from Nexus1) triggers <math>\overline{EVTO}</math></p> <p>XXXX1XXX Watchpoint #4 (DAC1 from Nexus1) triggers <math>\overline{EVTO}</math></p> <p>XXXXX1XX Watchpoint #5 (DAC2 from Nexus1) triggers <math>\overline{EVTO}</math></p>

*Note:* The EOC bits in DC1 must be programmed to trigger  $\overline{EVTO}$  on watchpoint occurrence for the EWC bits to have any effect.

**36.7.7.2 Development Status Register (DS)**

The development status register is used to report system debug status. When debug mode is entered or exited, or an e200z0h-defined low power mode is entered, a debug status message is transmitted with DS[31:24]. The external tool can read this register at any time.



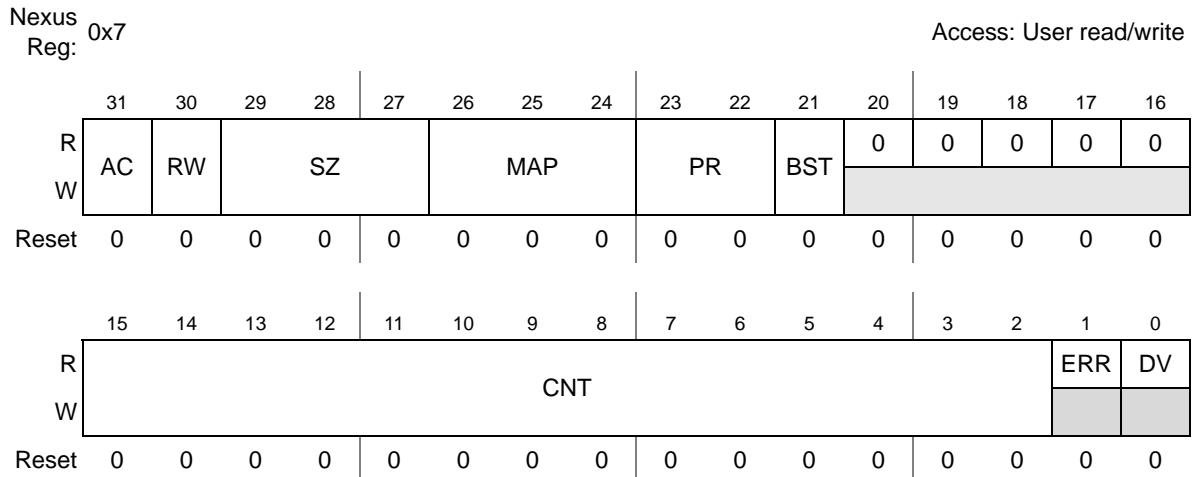
**Figure 586. Development Status Register (DS)**

**Table 560. DS Field Descriptions**

Field	Description
DBG	<p>e200z0h CPU debug mode status.</p> <p>0 CPU not in debug mode</p> <p>1 CPU in debug mode</p>
LPC[1:0]	<p>e200z0h CPU low power mode status.</p> <p>00 Normal (run) mode</p> <p>01 CPU in halted state</p> <p>10 CPU in stopped state</p> <p>11 Reserved</p>
CHK	<p>e200z0h CPU checkstop status.</p> <p>0 CPU not in checkstop state</p> <p>1 CPU in checkstop state</p>

### 36.7.7.3 Read/Write Access Control/Status (RWCS)

The read write access control/status register provides control for read/write access. Read/write access provides DMA-like access to memory-mapped resources on the system bus either while the processor is halted, or during runtime. The RWCS register also provides read/write access status information as shown in [Table 561](#).



**Figure 587. Read/Write Access Control/Status Register (RWCS)**

**Table 561. RWCS Field Description**

Field	Description
AC	Access control. 0 End access 1 Start access
RW	Read/write select. 0 Read access 1 Write access
SZ[2:0]	Word size. 000 8-bit (byte) 001 16-bit (half-word) 010 32-bit (word) 011 Reserved 100–111 Reserved (default to word)
MAP[2:0]	MAP select. 000 Primary memory map 001-111 Reserved
PR[1:0]	Read/write access priority. 00 Lowest access priority 01 Reserved (default to lowest priority) 10 Reserved (default to lowest priority) 11 Highest access priority

**Table 561. RWCS Field Description (continued)**

Field	Description
BST	Burst control. 0 Module accesses are single bus cycle at a time. 1 Module accesses are performed as burst operation.
CNT[13:0]	Access control count. Number of accesses of word size SZ
ERR	Read access error. See <a href="#">Table 562</a> .
DV	Read access data valid. See <a href="#">Table 562</a> .

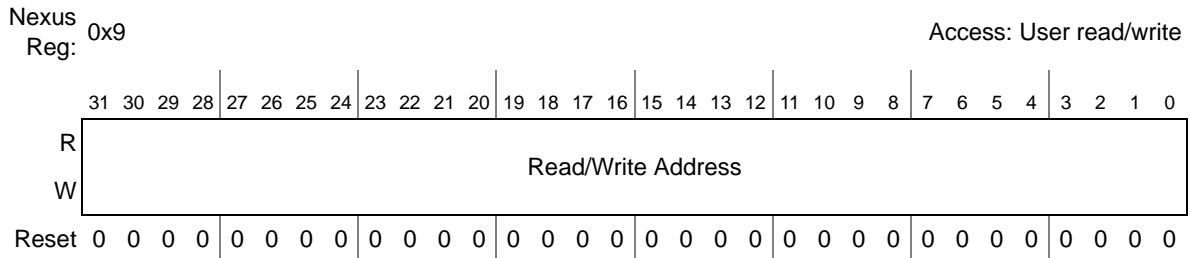
[Table 562](#) details the status bit encodings.

**Table 562. Read/Write Access Status Bit Encoding**

Read Action	Write Action	ERR	DV
Read access has not completed	Write access completed without error	0	0
Read access error has occurred	Write access error has occurred	1	0
Read access completed without error	Write access has not completed	0	1
Not allowed	Not allowed	1	1

**36.7.7.4 Read/Write Access Address (RWA)**

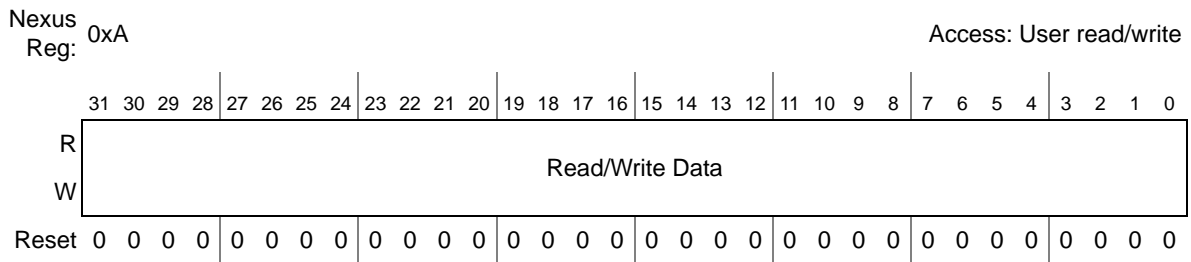
The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.



**Figure 588. Read/Write Access Address Register (RWA)**

**36.7.7.5 Read/Write Access Data (RWD)**

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.



**Figure 589. Read/Write Access Data Register (RWD)**

Table 563 shows the proper placement of data into the RWD. The “X” in the RWD column indicate byte lanes with valid data.

**Table 563. RWD Data Placement for Transfers**

Transfer Size and byte offset	RWA[2:0]	RWCS[SZ]	RWD			
			31:24	23:16	15:8	7:0
Byte	XXX	000	—	—	—	X
Half Word	XX0	001	—	—	X	X
Word	X00	010	X	X	X	X

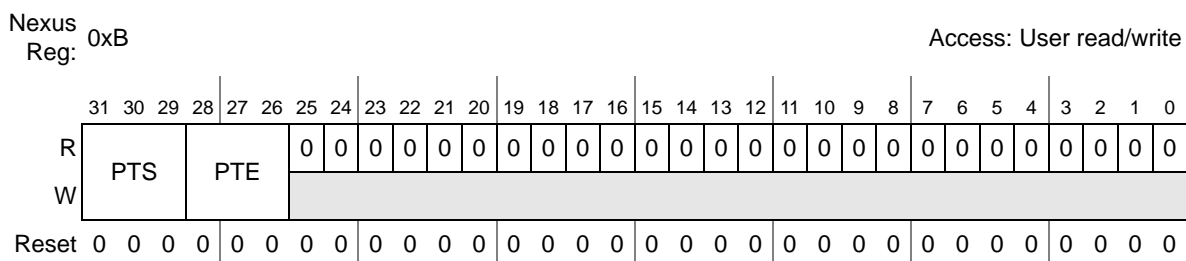
Table 564 shows the mapping of RWD bytes to byte lanes of the AHB read and write data buses.

**Table 564. RWD data placement for Transfers**

Transfer Size and byte offset	RWA[2:0]	RWD			
		31:24	23:16	15:8	7:0
Byte @000	000	—	—	—	AHB[7:0]
Byte @001	001	—	—	—	AHB[15:8]
Byte @010	010	—	—	—	AHB[23:16]
Byte @011	011	—	—	—	AHB[31:24]
Half@000	000	—	—	AHB[15:8]	AHB[7:0]
Half@010	010	-	-	AHB[31:24]	AHB[23:16]
Word@000	000	AHB[31:24]	AHB[23:16]	AHB[15:8]	AHB[7:0]

### 36.7.7.6 Watchpoint Trigger Register (WT)

The watchpoint trigger register allows the watch points defined within the e200z0h Nexus1 logic to trigger actions. These watch points can control program trace enable and disable. The WT bits can be used to produce an address related ‘window’ for triggering trace messages.



**Figure 590. Watchpoint Trigger Register (WT)**

Table 565 details the watchpoint trigger register fields.

**Table 565. WT Field Descriptions**

Field	Description
PTS[2:0]	Program trace start control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Reserved
PTE[2:0]	Program trace end control. 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1) 010 Use watchpoint #1 (IAC2 from Nexus1) 011 Use watchpoint #2 (IAC3 from Nexus1) 100 Use watchpoint #3 (IAC4 from Nexus1) 101 Use watchpoint #4 (DAC1 from Nexus1) 110 Use watchpoint #5 (DAC2 from Nexus1) 111 Reserved

*Note:* The WT bits control program trace only if the TM bits in the development control register 1 (DC1) have not already been set to enable program trace.

### 36.7.8 Nexus2+ Register Access via JTAG / OnCE

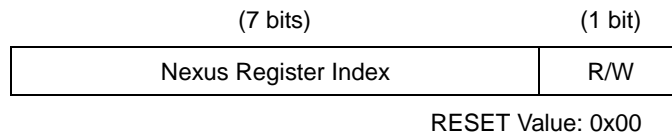
Access to Nexus2+ register resources is enabled by loading a single instruction (ACCESS\_AUX\_TAP\_ONCE) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS2\_ACCESS instruction (refer to Table 543). For the Nexus2+ module, the OCMD value is 0b00\_0111\_1100.

Once the ACCESS\_AUX\_TAP\_ONCE instruction has been loaded, the JTAG/OnCE port allows tool/target communications with all Nexus2+ registers according to the register map in Table 557.

Reading/writing of a Nexus2+ register then requires two (2) passes through the data-scan (DR) path of the JTAG state machine.

- The first pass through the DR selects the Nexus2+ register to be accessed by providing an index

(see [Table 557](#)), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG data register (DR). This register has the following format:



Nexus Register Index:	Selected from values in <a href="#">Table 557</a>
Read/Write (R/W):	0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, LSB first.
  - a) During a read access, data is latched from the selected Nexus register when the JTAG state machine passes through the CAPTURE-DR state.
  - b) During a write access, data is latched into the selected Nexus register when the JTAG state machine passes through the UPDATE-DR state.

### 36.8 Debug Implementation

This section describes the practical implementation of the debug port, its management, and the multiplexing strategy of its pads. It is required on SPC560B54/6x to support the Class 2+ (2003) debug features on the e200z0h.

### 36.9 Debug Capabilities

SPC560B54/6x supports the Class2+ debug features listed in [Table 566](#)

**Table 566. Nexus Debug Requirements**

Feature	Class 1	Class 2	Class 3 <sup>(1)</sup>	Class 4 <sup>(1)</sup>
Static debug	X	X	X	X
Set breakpoints/watchpoints	X	X	X	X
Watchpoint messaging	X	X	X	X
Ownership trace	—	X	X	X
Program trace (BTM)	—	X	X	X
Port replacement	—	optional	optional	optional
Data trace (write only)	—	—	X	X
Dynamic memory read/write	—	—	X	X
Data trace (read/write)	—	—	optional	optional
Data acquisition	—	—	optional	optional
Overrun Control	—	—	—	X



Table 566. Nexus Debug Requirements (continued)

Feature	Class 1	Class 2	Class 3 <sup>(1)</sup>	Class 4 <sup>(1)</sup>
Memory substitution	—	—	—	X
Complex triggering	—	—	—	X
External memory substitution	—	—	—	optional

1. Nexus 2+ has some class 3 and some class 4 features.

## 37 Revision History

[Table 567](#) summarizes revisions to this document.

**Table 567. Document revision history**

Date	Revision	Changes
25-Nov-2008	1	Initial release (internal only)
28-Apr-2009	2	<p>Editorial and formatting changes throughout reference manual</p> <p><i>Chapter 1, "Overview"</i>: Changed title from "Introduction" to "Overview"</p> <p>Updated <i>Section 1.1, "Introduction"</i></p> <p>Updated <i>Table 1</i></p> <p><i>Figure 1</i>: Added STM; changed SIU to SIUL</p> <p><i>Section 1.4, "Feature Summary"</i>: Harmonized oscillator naming</p> <p><i>Section 1.4.5, "Flash Memory"</i> feature "Error correction status": Removed redundant bullet "Flash ECC single-bit errors reported"</p> <p><i>Section 1.4.8, "Boot Assist Module (BAM)"</i>: Edited description</p> <p><i>Section 1.4.16, "On-chip voltage regulator (VREG)"</i>: removed note in brackets</p> <p><i>Section 1.4.17, "Analog to Digital Converter Module (ADC)"</i>: replaced "4 configurable analog comparator channels" with "Up to 6 configurable analog comparator channels" in bullet list</p> <p>Added <i>Section 1.6, "Memory Map"</i> (was previously separate chapter)</p> <p><i>Table 5</i>:</p> <ul style="list-style-type: none"> <li>– Removed footnote number from LINFlex 8 and LINFlex 9</li> <li>– Removed "Code Flash Array 1 Test Sector" and "Code Flash Array 2 Test Sector" rows</li> <li>– Changed (PIT/RTI) to (PIT)</li> </ul> <p><i>Section 0.0.21, "Nexus Port Controller (NPC)"</i>: Removed bullet about 2 and 4 MDO</p> <p><i>Chapter 2, "Signal Description"</i>:</p> <p>Updated <i>Section 2.1, "Package Pinouts"</i>: Repaired cross-reference</p> <p>Updated <i>2.2, "Pad configuration during reset phases"</i></p> <p>Added <i>2.3, "Pad configuration during standby mode exit"</i></p> <p>Updated <i>Table 7</i></p> <p><i>2.5, "Pad types"</i>: Updated second footnote</p> <p><i>Table 9</i>: Updated table and footnotes</p> <p><i>Table 10</i>: Inserted overbar on port pins EVTI, EVTO and MSEO</p> <p><i>Section 0.2, "Pad configuration during standby mode exit"</i>: Added two footnotes</p> <p><i>Table 2</i>: Replaced the RESET config. of RESET with "Input, weak pull-up only after PHASE2"</p> <p><i>Chapter 3, "Reset Generation Module (RGM)"</i>: Major update of entire chapter</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
28-Apr-2009	2	<p><i>Chapter 4, "Clock Description"</i>: Major update of entire chapter</p> <p><i>Chapter 5, "Mode Entry Module (ME)"</i>:  Updated <i>Section 5.4.2, "Modes Details"</i> and <i>Section 5.4.3, "Mode Transition Process"</i>  — Corrected typos; added wakeup alarm list; expanded list of modules still powered during STANDBY0; added <i>Table 69</i> and <i>Table 70</i>; added customer warning about illegal Flash power mode switching</p> <p><i>Chapter 6, "e200z0 Core"</i>: Amended and aligned content to e200z0h core; removed e200z1 content</p> <p><i>Chapter 7, "Enhanced Direct Memory Access (eDMA)"</i>:  Added <i>Section 7.1, "Information specific to this device"</i>  Changed register bit numbering from LSB = 0 to MSB = 0</p> <p>Added <i>Chapter 8, "DMA Channel Multiplexer (DMA_MUX)"</i></p> <p><i>Chapter 9, "Multi-Layer AHB Crossbar Switch (XBAR)"</i>: Added <i>Section 9.1, "Information specific to this device"</i></p> <p><i>Chapter 10, "Peripheral Bridge (PBRIDGE)"</i>: Updated names of PBRIDGE1 peripherals in <i>Figure 121</i></p> <p><i>Chapter 11, "Flash Memory"</i>:  Updated <i>Table 118</i>  <i>Section 11.4.1, "Introduction"</i>: Replaced "SPP" with "RPP"  <i>Section 11.4.1.2, "Features"</i>: Removed references to figure "Flash Memory Controller Block Diagram" (figure does not exist)  <i>Table 218</i>: Updated reset values and removed footnotes  <i>Figure 134</i>: Added "Read, no buffering, no prefetch, APC = 0, RWSC = 0, PFLM = 0" to top of diagram  Replaced "Macrocell" with "module" throughout document  <i>Section 18.2.4.3, "User Mode Operation"</i>: Editorial changes  <i>Section 18.2.4.5, "Power-down mode"</i>: Editorial changes  <i>Section 18.2.4.6, "Low Power Mode"</i>: Editorial changes  <i>Section 18.3.5.2, "Power-Down mode"</i>: Editorial changes  <i>Section 18.3.5.3, "Low Power Mode"</i>: Editorial changes</p> <p><i>Chapter 12, "General-Purpose Static RAM (SRAM)"</i>: Editorial and formatting changes</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
28-Apr-2009	2	<p><i>Chapter 13, "Memory protection unit (MPU)":</i></p> <p><i>Table 234:</i></p> <ul style="list-style-type: none"> <li>– Updated reset values</li> <li>– Removed MPU_EAR3 and MPU_EDR3</li> </ul> <p>Updated <i>Section 13.2.2, "Register description"</i> to include adding bit numbers to field names and changing field bit numbers format to LSB=0 where needed</p> <p><i>Figure 134:</i> Replaced asterisks with '0' as reset value for fields HRL[3:0], NSP[3:0] and NRGD[3:0]</p> <p><i>Figure 157:</i> Removed MPU_EAR3 content</p> <p><i>Figure 158:</i> Removed MPU_EDR3 content</p> <p><i>Figure 137:</i> Replaced asterisks with '0' as reset value for bits 27:31</p> <p><i>Table 192:</i> Replaced SRTADDR[31:0] with SRTADDR[26:0]</p> <p><i>Table 193:</i> Replaced ENDADDR[31:0] with ENDADDR[26:0]</p> <p><i>Chapter 14, "Interrupt Controller (INTC)":</i> Major update of entire chapter</p> <p><i>Chapter 15, "System Integration Unit Lite (SIUL)":</i> Major update of entire chapter</p> <p><i>Chapter 16, "Error Correction Status Module (ECSM)":</i></p> <p>Editorial and formatting changes throughout</p> <p><i>Table 284:</i> Removed MRSR register and described as reserved</p> <p><i>Section 16.4.2, "Register description":</i> Removed MRSR register</p> <p><i>Section 16.4.2.6, "Miscellaneous User-Defined Control Register (MUDCR)":</i> Updated text following field description table</p> <p><i>Section 16.4.2.7.3, "ECC Error Generation Register (EEGR)":</i> Replaced "for the ECC Configuration Register definition" with "for the ECC Error Generation Register definition" in sentence above bitmap</p> <p><i>Section 16.4.2.8.3, "RAM ECC Syndrome Register (RESR)":</i> Repaired cross-reference links to <i>Table 300</i></p> <p>Removed section "High Priority Enables"</p> <p><i>Chapter 17, "System Timer Module (STM)":</i> None</p> <p><i>Chapter 18, "Real Time Clock / Autonomous Periodic Interrupt (RTC/API)":</i> Updated and harmonized oscillator naming</p> <p><i>Chapter 19, "Boot Assist Module (BAM)":</i> Major update of entire chapter</p> <p><i>Chapter 20, "Configurable Enhanced Modular IO Subsystem (eMIOS)":</i> Major update of entire chapter</p> <p><i>Chapter 21, "Wakeup Unit (WKPU)":</i> Major update of entire chapter</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
28-Apr-2009	2	<p><i>Chapter 22, "System Status and Configuration Module (SSCM)":</i>                      Made minor editing and formatting changes to improve readability  <i>Section 22.2.2, "Register description":</i> Updated introduction and applied LSB=0 numbering to field bit numbers where needed                      Updated <i>Section 22.2.2.1, "System Status Register (STATUS)"</i>                      Updated <i>Section 22.2.2.2, "System Memory Configuration Register (MEMCONFIG)"</i>  <i>Section 22.2.2.3, "Error Configuration (ERROR)":</i> Updated description of field RAE                      Removed section "Initialization/Application Information"</p> <p><i>Chapter 23, "Analog-to-Digital Converter (ADC)":</i> Major update of entire chapter</p> <p><i>Chapter 24, "Safety":</i>                      Editorial and formatting changes                      Replaced PBRIDGE with AIPS                      Major update of <i>Section 24.2, "System Watchdog Timer (SWT)"</i></p> <p><i>Chapter 25, "Deserial Serial Peripheral Interface (DSPI)":</i> Major update of entire chapter</p> <p><i>Chapter 26, "FlexCAN Module":</i>                      Editorial and formatting changes                      Added <i>Section 26.1, "Information Specific to This Device"</i>  <i>Figure 349:</i> Reformatted field bit numbers to MSB=0                      Added <i>Section 26.5.2, "Local Priority Transmission"</i></p> <p><i>Chapter 27, "CAN Sampler":</i>                      Minor editing and formatting changes                      Updated oscillator naming for 16 MHz fast internal RC oscillator</p> <p><i>Chapter 28, "Inter-Integrated Circuit Bus Controller Module (I2C)"</i>  <i>Section 28.1.1, "Overview":</i> Changed capacitance value from "400pF" to "50pF"  <i>Table 482:</i> Removed Mode column and footnotes</p> <p><i>Chapter 29, "Periodic Interrupt Timer (PIT)":</i>                      Editorial and formatting changes                      Replaced PIT_RTI with PIT throughout document                      Corrected number of timer channels: was 4, is 8  <i>Table 571:</i> Added base address                      Updated <i>Figure 386</i>                      Updated <i>Table 493</i>                      Updated <i>Table 494</i>  <i>Section 29.5.1, "Example configuration":</i> Removed RTI lines from code</p>

Table 567. Document revision history (continued)

Date	Revision	Changes
28-Apr-2009	2	<p><i>Chapter 30, "Cross Triggering Unit (CTU)":</i>            Editorial and formatting changes  <i>Table 501:</i>            – Added base address            – Added page index column            Amended end of <i>Section 26.4.2, "Event Configuration Register (CTU_EVTTCFGRx) (x = 0...63)"</i>            Updated <i>Section 30.5, "Functional description"</i>            Added <i>Table 504</i></p> <p><i>Chapter 31, "LIN Controller (LINFlex)":</i> Major update of entire chapter</p> <p><i>Chapter 8, "Power Control Unit (PCU)":</i> Major update of entire chapter</p> <p><i>Chapter 33, "Voltage Regulators and Power Supplies":</i>            Replaced all "High Power or Main Regulator (HPREG)" with "High Power Regulator (HPREG)"            Modified <i>Section 33.1.4, "LVDs and POR"</i>            Updated <i>Figure 408</i></p> <p><i>Chapter 34, "IEEE 1149.1 Test Access Port Controller (JTAGC)":</i> Major update of entire chapter</p> <p><i>Chapter 35, "Nexus Port Controller (NPC)":</i>            Added NEXCFG bit in NPC_PCR[23] and corresponding nex_nexcfg output port</p> <p><i>Appendix A, "Register Under Protection":</i> None</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
16-Oct-2009	3	<p><i>Chapter 1 Overview</i>                      Updated SPC560B54/6x Device comparison                      Updated Block diagram                      Updated Feature summary                      Updated ADC and LINFlex features</p> <p><i>Chapter 2 Signal description</i>                      Updated EXTAL function description                      Replaced Alternate Function AF with ALT globally in the document                      Updated port pin PA[2], PA[4], PA[6], PA[9], PA[12], PA[3], PB[0] and PB[1]                      Created mapping for 8 additional ADC on PA[3], PA[7], PA[10], PA[11], PB[8], PB[9], PB[10] and PE[12]                      Replaced all hyphen with em-dashes                      Updated Package Pinouts</p> <p><i>Chapter 3 Clock description</i>                      Updated Main features                      Updated Clock monitor unit diagram</p> <p><i>Chapter 4 Clock Generation Module (MC_CGM)</i>                      Added 'Location' column to MC_CGM Register description; adjusted all tables and figures to JDP standard width; added clock domain information to clock source selection register descriptions</p> <p><i>Chapter 5 Mode Entry Module (MC_ME)</i>                      Added 'Location' column to MC_ME Register description; added note for S_MTRANS polling; cleaned up MC_ME Mode diagram; added details to RESET mode description; added details of booting from backup RAM on STANDBY0 exit</p> <p><i>Chapter 6 Boot Assist Module (BAM)</i>                      Updated Download and execute the new code                      Updated Download 64-bit password and password check                      Updated Password check flow diagram</p> <p><i>Chapter 7 Reset Generation Module (MC_RGM)</i>                      Added 'Location' column to MC_RGM Register description; adjusted all tables and figures to JDP standard width</p> <p><i>Chapter 9 Power Control Unit (MC_PCU)</i>                      Added 'Location' column to MC_PCU Memory Map; adjusted all tables and figures to JDP standard width</p> <p><i>Chapter 12 Enhanced Direct Memory Access (eDMA)</i>                      Added SPC560B54/6x device specific information -16DMA channels with no groups and no arbitration</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
16-Oct-2009	3	<p><i>Chapter 13 DMA Channel Multiplexer (DMA_MUX)</i>  Updated DMA_MUX channel mapping table  Updated number of Peripheral slots  Updated Example 3: Switch DMA Channel 8 from source #5 transmit to source #7 transmit</p> <p><i>Chapter 15 Memory Protection Unit (MPU)</i>  Updated MPU block diagram  —Replaced Zen Core (z0hn1 or z0hn2p) to e200z0h core  —Replaced AXBS with XBAR</p> <p><i>Chapter 18 IEEE 1149.1 Test Access Port Controller (JTAGC)</i>  Updated Part identification number for SPC560B54/6x in Device identification register field description</p> <p><i>Chapter 21 Flash memory</i>  Replaced AXBS-lite with XBAR across the document  Updated Code Flash block-User mode operation  Updated MCR-reset values for SIZE 1, SIZE2 and DONE  Updated the following for both Code Flash memory and data Flash memory  Updated Functional description-Module structure  Updated Reset  Updated Test Flash block-User mode operation  Updated MCR-reset values for SIZE 1, SIZE2 and DONE  Updated programming considerations-Double word program  Updated MCR fields descriptions: Field 21, 22, Array Space Size  Updated UT0 fields descriptions: Field 26, 28,29  Updated Programming considerations-Modify operation  Updated User test mode-Array integrity self check, Margin read  Updated Error correction code- added EEprom emulation</p> <p><i>Chapter 23 LIN Controller (LINFlexD)</i></p> <p><i>Chapter 22 LIN Controller (LINFlex)</i>  Included both LINFlex3 and LINFlex2 chapters</p> <p><i>Chapter 24 FlexCAN module</i>  FlexCAN Memory Mapping: Added Location column  Rx FIFO Structure: Editing updates</p> <p><i>Chapter 25 CAN Sampler</i>  Minor editing and formatting changes</p> <p><i>Chapter 26 Inter-Integrated Circuit Bus Controller Module (I2C)</i>  Added DMA Interface section</p>



**Table 567. Document revision history (continued)**

Date	Revision	Changes
16-Oct-2009	3	<p><i>Chapter 27 Configurable Enhanced Modular IO Subsystem (eMIOS)</i>                      Added Center Aligned Output Pulse Width Modulation Buffered                      Removed channel Type F from Channel Configuration diagram                      Removed Figure 254-Unified Channel Block and added reference to Unified Channel block diagram                      Removed Note from eMIOS memory map                      Removed Note from eMIOS Output Update Disable (EMIOSOUDIS) Register                      Removed Note from eMIOS Disable Channel (EMIOSUCDIS) Register                      Updated eMIOS UC Counter (EMIOSCNT[n]) Register                      Updated Functional description</p> <p><i>Chapter 28 Analog-to-Digital Converter (ADC)</i>                      Updated ADC implementation diagram for SPC560B54/6x                      Updated Device specific features to include 8 additional 12-bit medium accuracy channels information                      Updated Injected channel conversion section</p> <p><i>Chapter 32 Wakeup Unit (WKPU)</i>                      SPC560B54/6x Device Specific updates                      Created WKUP line mapping for GPIO[103], GPIO[105], GPIO[89] and GPIO[131]</p> <p><i>Chapter 34 System Timer Module (STM)</i>                      Added a STM counter stop note after Functional description</p> <p>Chapter 36, "Nexus Port Controller (NPC)"                      Added SPC560B54/6x device specific header</p>
23-Nov-2009	4	<p><i>Chapter 1 Overview</i>                      Updated 1.5M Device comparison table</p>

Table 567. Document revision history (continued)

Date	Revision	Changes
01-Feb-2010	5	<p><a href="#">Chapter 1 Overview</a> Updated block diagram to show pad connections</p> <p><a href="#">Chapter 2 Signal description</a> Updated Functional port pin descriptions table</p> <p><a href="#">Chapter 3 Clock description</a> Updated system clock generation diagram</p> <p><a href="#">Chapter 11 e200z0h Core</a> Updated block diagram to remove MPU from core complex</p> <p><a href="#">Chapter 21 Flash memory</a> Updated UT0 field description Updated ECC logic check</p> <p><a href="#">Chapter 23 LIN Controller (LINFlexD)</a> Updated Error calculation for programmed baud rate table and LFDIV Note above it Updated IFMI bit in IFMI register</p> <p><a href="#">Chapter 30 Safety</a> Included KEY at bit 22 in SWT_CR register</p> <p><a href="#">Chapter 32 Wakeup Unit (WKPU)</a> Included note about Wakeup pins at External Signal Description</p> <p><a href="#">Chapter 36 Voltage Regulators and Power Supplies</a> Updated Voltage Regulators (HPREG and LPREG) section</p>
06-Sep-2010	6	<p>Replaced instances of “02 Sep 2009” with “16 Oct 2009” as correct date in Rev. 3 revision history; reorganized chronology of changes.</p> <p>Editorial updates</p> <p><a href="#">Chapter 1 Overview</a> In the “SPC560B54/6x family comparison” figure: – Replaced “RAM” with “SRAM” – Replaced “DMA” with “eDMA” In the “SPC560B54/6x block diagram” figure: – Replaced “CAN” with “FlexCAN” in the legend – Replaced “RAM” with “SRAM” – Added the “5ch 12-bit ADC” block – Added “eDMA” in the legend – Improved visual quality – Removed the Nexus row from the LEGEND because it is not a abbreviation. In the “Analog-to-Digital Converter (ADC)” section: – Updated the total number of the single ended input channels (it was 48) and thus the total number of the channels also considering the expandable ones (it was 92). – Changed the specifications about the shared and dedicated channels as for 10-bit and 12-bit ADC. In the “Memory Protection Unit (MPU)” section: – Rephrased the introduction.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
06-Sep-2010	6 (cont.)	<p><i>Chapter 2 Signal description</i></p> <p>Deleted muxing details from the LQFP diagrams (port pins remain).                      Added cross-references to the port function table which shows muxing detail.                      Updated Section “Pad configuration during standby mode exit” with more information on TDO pin floating in STANDBY mode.                      208 BGA package:                      – XTAL at R9 changed to XTAL32                      – EXTAL at T9 changed to EXTAL32                      Section “Pad configuration during reset phases” replaced “RESET pad is driven low. This is released only after PHASE2 reset completion” with “RESET pad is driven low by the device till 40 FIRC clock cycles after phase2 completion. Minimum phase3 duration is 40 FIRC cycles.”                      Replaced Section “Pad configuration during standby mode exit”.                      Section “Voltage supply pins”                      – Changed “two” to “three” dedicated pins are used for 1.2 V regulator stabilization                      – Corrected pin information in “Voltage supply pin descriptions” table                      – Corrected pin information in System pin descriptions table.                      Multiple changes in Functional port pin descriptions table.</p> <p><i>Chapter 3 Clock description</i></p> <p>Editorial and formatting changes                      Added Progressive clock switching scheme figure                      Update definition of en_pll_sw bit filed on Control Register.                      Revised “Progressive clock switching” section.                      Fast external crystal oscillator digital interface (FXOSC):                      – Rephrased “Main features” section                      – Replaced “CK_OSCM” with “FXOSC” and “OSC MODE” with “Oscillator Mode in the “Truth table of crystal oscillator” table.                      Slow external crystal oscillator digital interface (SXOSC):                      – Rephrased “Main features” section                      – Replaced “CK_OSCM” with “SXOSC” and “OSC MODE” with “Oscillator Mode in the “Truth table of crystal oscillator” table.                      – Added paragraph about the two’s complement trimming method.                      Fast internal RC oscillator (FIRC) digital interface:                      – Added paragraph about the two’s complement trimming method.                      – Added “FIRCON_STDBY” field in the “RC Oscillator Control Register (RC_CTL)” register                      – Update the “RC Oscillator Control Register (RC_CTL) filed description” table                      Frequency-modulated phase-locked loop (FMPLL):                      – 1:1 mode is not longer applicable. Affected sections are: Note in the FMPLL block diagram, FMPLL feature list, FMPLL Control register figure, CR Field description table, 1:1 mode section                      – Replaced “Reduced frequency divider (RFD)” with “Frequency divider (FD)”                      – Replaced “SSSCG_EN” with “FM_EN”                      – Updated “Recommendations” section</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
06-Sep-2010	6 (cont.)	<p><i>Chapter 3 Clock description</i> (continued)                      Clock monitor unit (CMU):</p> <ul style="list-style-type: none"> <li>– Updated "CMU block diagram"</li> <li>– Updated "Main features" section</li> <li>– Replaced "FOSC" and "XOSC" with "FXOSC" on "Clock monitor unit diagram" figure</li> <li>– Updated "FMPLL clock monitor" section.</li> </ul> <p><i>Chapter 6 Boot Assist Module (BAM)</i>                      Added notes in the following sections:</p> <ul style="list-style-type: none"> <li>– Download 64-bit password and password check</li> <li>– Download data</li> <li>– Execute code</li> </ul> <p><i>Chapter 7 Reset Generation Module (MC_RGM)</i>                      Added <i>Figure 76: Reset configuration timing</i></p> <p><i>Chapter 8 System Integration Unit Lite (SIUL)</i></p> <ul style="list-style-type: none"> <li>– Clarified description of I/O pad function in overview section.</li> <li>– Clarification: Not all GPIO pins have both input and output functions.</li> <li>– Replaced parallel port register sections (PGPDO, PGPD, and MPD), clarifying register function and bit ordering.</li> <li>– Editorial updates throughout chapter.</li> </ul> <p><i>Chapter 10 Interrupt Controller (INTC)</i>                      Updated <i>Table 112: Interrupt vector table</i>: the IRQs No.: 9, 67, 107, 175, 192 and 204</p> <p><i>Chapter 12 Enhanced Direct Memory Access (eDMA)</i>                      General updates:</p> <ul style="list-style-type: none"> <li>– Most text in this chapter has been rewritten for clarity.</li> </ul> <p>Updates to features list:</p> <ul style="list-style-type: none"> <li>– LinFlexD added to list of peripheral DMA request sources.</li> <li>– 'C' pseudocode specification of TCD deleted.</li> </ul> <p>Updates to Memory map/register definition</p> <ul style="list-style-type: none"> <li>– All registers have been renamed. Details are below.</li> <li>– Previously, eDMA controller was documented generically, showing support for up to 64 channels. Registers changed to match implementation of 16 channels. Several registers are shown as 32-bit registers even though the most significant 16-bits are reserved.</li> </ul>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
06-Sep-2010	6 (cont.)	<p><i>Chapter 12 Enhanced Direct Memory Access (eDMA)</i> (continued)</p> <p>Updates to registers:</p> <ul style="list-style-type: none"> <li>– DMACR[GRP3PRI] field deleted</li> <li>– DMACR[GRP2PRI] field deleted</li> <li>– DMACR[GRP1PRI] field deleted</li> <li>– DMACR renamed to EDMA_CR</li> <li>– DMAES[ECX] field deleted</li> <li>– DMAES[GPE] field deleted</li> <li>– DMAES renamed to EDMA_ESR</li> <li>– DMAERQH register deleted</li> <li>– DMAERQL register renamed EDMA_ERQRL</li> <li>– DMAEEIH register deleted</li> <li>– DMAEEIL register renamed to EDMA_EEIRL</li> <li>– DMACEEI[NOP] field deleted</li> <li>– DMACEEI register renamed to EDMA_CEEIR</li> <li>– DMASEEI[NOP] field deleted</li> <li>– DMASEEI register renamed to EDMA_SEEIR</li> <li>– DMACERQ[NOP] field deleted</li> <li>– DMACERQ register renamed to EDMA_CERQR</li> <li>– DMASERQ[NOP] field deleted</li> <li>– DMASERQ register renamed to EDMA_SERQR</li> <li>– DMASRT[NOP] field deleted</li> <li>– DMASRT register renamed to EDMA_SSBR</li> <li>– DMACDNE[NOP] field deleted</li> <li>– DMACDNE register renamed to EDMA_CDSBR</li> <li>– DMACERR[NOP] field deleted</li> <li>– DMACERR register renamed to EDMA_CER</li> <li>– DMACINT[NOP] field deleted</li> <li>– DMACINT register renamed to EDMA_CIRQR</li> <li>– DMAINTH register deleted</li> <li>– DMAINTL register renamed to EDMA_IRQRL</li> <li>– DMAERRH register deleted</li> <li>– DMAERRL register renamed to EDMA_ERL</li> <li>– DMAHRSH register deleted</li> <li>– DMAHRSL register renamed to EDMA_HRSL</li> <li>– DMAGPOR register deleted</li> <li>– DCHPRix registers renamed to EDMACPRx</li> </ul> <p><i>Chapter 15 Memory Protection Unit (MPU)</i> Updated <i>Figure 144: MPU block diagram</i></p> <p><i>Chapter 18 IEEE 1149.1 Test Access Port Controller (JTAGC)</i> Added a paragraph into “External Signal Description” that explain when the device get incompliance with IEEE 1149.1-2001. Changed the code values for ACCESS_AUX_TAP_TCU and ACCESS_AUX_TAP_NPC in the “JTAG Instructions” table.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
06-Sep-2010	6 (cont.)	<p><i>Chapter 20 Deserial Serial Peripheral Interface (DSPI)</i> Included Bit fields CLR_TXF and CLR_RXF in DSPIx_MCR register</p> <p><i>Chapter 21 LIN Controller (LINFlex)</i> <i>Chapter 21.2.1 LIN mode features:</i> Added the “Peripheral DMA request sources“ item.</p> <p><i>Chapter 26 Analog-to-Digital Converter (ADC)</i> <i>Section 26.1.1 Device-specific features:</i> Updated the total number of the single ended input channels (it was 48) and thus the total number of the channels also considering the expandable ones (it was 92). Changed the specifications about the shared and dedicated channels as for 10-bit and 12-bit ADC. Added <i>Figure 365: Sampling and conversion timings</i> Added <i>Table 364: Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V for ADC_0</i> Updated:  <ul style="list-style-type: none"> <li>– “Overview” section</li> <li>– “Introduction” section</li> <li>– “Injected channel conversion” section</li> <li>– “Abort conversion” section</li> <li>– ADC CTU (Cross Triggering Unit)</li> <li>– Presampling section</li> <li>– “12-bit ADC_a digital registers” table</li> </ul>                     Updated following registers:  <ul style="list-style-type: none"> <li>– CEOCFR</li> <li>– CIMR</li> <li>– WTISR</li> <li>– DMAR</li> <li>– PSR</li> <li>– NCMR</li> <li>– JCMR</li> <li>– CDR</li> <li>– CWSEL</li> <li>– CWENR</li> <li>– AWORR</li> </ul>                     Inserted “CTU triggered conversion” in the conversion list of “Functional description” section                      Replaced generic “system clock” with “peripheral set 3 clock”                      “ADC sampling and conversion timing” section, added information about “ADC_1”                      Moved CWSEL, CWENR and AWORR register to “Watchdog register“ section                      CTR register: Inserted a footnote about OFFSHIFT field stating: “available only for CTR0”</p> <p><i>Chapter 30 Safety</i> Added note indicating Watchdog performance during BAM execution.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
06-Sep-2010	6 (cont.)	<p><i>Chapter 32 Wakeup Unit (WKPU)</i></p> <p><i>Section 32.1, "Overview":</i> Updated the interrupt vectors</p> <p>Updated <i>Figure 553: Wakeup unit block diagram</i></p>
19-Jul-2011	7	<p>Editorial changes and improvements (including reformatting of memory maps, register figures, and field descriptions to a consistent format) changes throughout reference manual. Rearranged the chapter order.</p> <p>Added Chapter Preface</p> <p>Chapter Introduction:</p> <p>Changed the chapter title (was "Overview", is "Introduction").</p> <p>Renamed "Introduction" to "The SPC560B54/6x microcontroller family".</p> <p>In the device-comparison table, changed "32 kHz" to "32 KHz".</p> <p>Renamed "Feature summary" to "Feature details" and revised it as follows:</p> <ul style="list-style-type: none"> <li>– Removed second level bullet from "Interrupt Controller (INTC)" section.</li> <li>– Removed "Flash partitioning" table.</li> <li>– Rewrote first bullet within "Cross Trigger Unit (CTU)" section.</li> <li>– Updated the "Serial Communication Interface Module (LINFlex)" section.</li> <li>– In the "System timers" section, corrected name of PIT (was Peripheral Interrupt Timer; is Periodic Interrupt Timer).</li> <li>– In the "System timers" section, changed "32 kHz" to "32 KHz".</li> </ul> <p>In the block diagram:</p> <ul style="list-style-type: none"> <li>– Added ECSM, GPIO and VREG to the legend.</li> <li>– Added "Interrupt request with wakeup functionality" as an input to the WKPU block.</li> </ul> <p>Moved the "Memory map" section to its own separate chapter.</p> <p>Chapter Memory Map</p> <p>Added this chapter (content previously contained in the Overview chapter).</p> <p>Changed "Test sector data flash memory array 0" to "Data flash memory array 0 test sector".</p> <p>Changed the reserved entry at 0x4000_8000 (is now at 0x4001_8000).</p> <p>Changed the entries at 0xC3FB_0000 and 0xC3FB_4000 (are reserved).</p> <p>Revised the reserved entry at 0xFFE0_C000 (is now at 0xFFE0_8000, with size 160 KB).</p> <p>Deleted "(PBRIDGE_1)" and "(PBRIDGE_0)".</p> <p>Consolidated multiple adjacent reserved rows into single rows.</p> <p>Chapter Signal Description</p> <p>Renamed "Functional ports A, B, C, D, E, F, G, H, I, J" to "Functional port pins".</p> <p>In the "Functional port pin descriptions" table:</p> <ul style="list-style-type: none"> <li>– For PB[8] and PB[9], added the footnote "This wakeup input cannot be used to exit STANDBY mode" to the WKPU functions.</li> <li>– Added a cross-reference to the footnote "All WKPU pins also support external interrupt capability..." from pins WKPU[25:28].</li> </ul> <p>In the "Functional port pin descriptions" table, changed the I/O direction for the following from "—" to "I/O":</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>In the “Functional port pin descriptions” table, changed the I/O direction for the following from “O” to “I/O”:</p> <ul style="list-style-type: none"> <li>– PF[2], CS0_2</li> <li>– PG[3], CS0_3</li> <li>– PG[11], CS0_4</li> </ul> <p>In the “Functional port pin descriptions” table, changed the I/O direction for the following from “I/O” to “O”:</p> <ul style="list-style-type: none"> <li>– PH[12], CS1_4</li> <li>– PH[14], CS1_3</li> <li>– PI[7], CS1_4</li> <li>– PI[13], CS1_3</li> <li>– PJ[0], CS1_4</li> <li>– PJ[3], CS1_5</li> </ul> <p>For PH[13], changed the I/O direction for AF3 (was —, is I/O).</p> <p>Chapter Safety Migrated the chapter contents to the “Register Protection” and “SWT” chapters.</p> <p>Added Chapter Microcontroller Boot</p> <p>Chapter Clock Description Restored the system clock generation figure to the version that existed in Rev. 5 of this reference manual, and then changed “32 kHz” to “32 KHz”. (The change to this figure in Rev. 6 was not intentional.)</p> <p>Fast external crystal oscillator (FXOSC) digital interface section: Changed the sentence from “The FXOSC digital interface controls the 4–40 MHz fast external crystal oscillator (FXOSC).” to “The FXOSC digital interface controls the operation of the 4–40 MHz fast external crystal oscillator (FXOSC).”</p> <p>Truth table of crystal oscillator table: Replaced “ME_GS.S_XOSC” with “ME_xxx_MC[FXOSCON]”, replaced “FXOSC_CTL.OSCBYP” with “FXOSC_CTL[OSCBYP]”</p> <p>Slow external crystal oscillator (SXOSC) digital interface section: Changed the sentence from “The SXOSC digital interface controls the 32 KHz slow external crystal oscillator (SXOSC).” to “The SXOSC digital interface controls the operation of the 32 KHz slow external crystal oscillator (SXOSC).”</p> <p>SXOSC truth table: Replaced “S_OSC” with “OSCON”</p> <p>Renamed the figure title from “RC Oscillator Control Register (RC_CTL)” to “FIRC Oscillator Control Register (FIRC_CTL)”</p> <p>Renamed the table title from “RC Oscillator Control Register (RC_CTL) field descriptions” to “FIRC Oscillator Control Register (FIRC_CTL) field descriptions”</p> <p>In the FXOSC_CTL figure, added footnotes to clarify the access to the OSCBYP and _OSC fields.</p> <p>Deleted the “CMU register map” section.</p>



**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>In the CGM_SC_SS section, added information about the clocks affected by the register.</p> <p>Chapter Mode Entry Module                      Changed WARNING to CAUTION.                      Changed HALT0 to HALT.                      Changed STOP0 to STOP.                      Changed STANDBY0 to STANDBY.                      In the “Mode transition process” section, added text explaining how to switch on the FXOSC.                      In the ME_&lt;mode&gt;_MC[DFLAON] field description, added a note about configuring reset sources as long resets.</p> <p>Chapter Reset Generation Module                      Changed STANDBY0 to STANDBY.                      Changed the RGM_FERD[D_EXR] field from read-only to read/write.                      Revised the RGM_DERD section to indicate that the register is always read-only.                      Revised the RGM_FEAR[AR_CMU_OLR] field description.                      Revised the RGM_DEAR section to indicate that the register is always read-only.                      Revised the RGM_FES[F_CORE] field description.                      Changed “core reset” to “debug control core reset”.</p> <p>Chapter Power Control Unit                      Changed HALT0 to HALT.                      Changed STOP0 to STOP.                      Changed STANDBY0 to STANDBY.                      In the memory map and PCU_PCONF2...3 section, split the single register figure into two separate figures for PCU_PCONF2 and PCU_PCONF3 to show the difference between the two.                      Added the “Initialization information” and “Application information” sections.</p> <p>Chapter Voltage Regulators and Power Supplies                      In the “Register description” section, added information on where to find the VREG_CTL base address.                      Revised the “Register description” section to include the address offset and MC_PCU mapping.</p> <p>Chapter Wakeup Unit                      Changed WKUP to WKPU to match the official module abbreviation.                      In the Overview section, changed “2 internal sources and up to 23 external sources” to “2 internal sources and up to 27 external sources”.                      In the Overview section, added a footnote to PB[8]... and PB[9]... indicating that these lines cannot exit STANDBY mode.                      In the “NMI management” section, changed “This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.” to “The NIF and NOV fields in this register are cleared by writing a ‘1’ to them; this prevents inadvertent overwriting of other flags in the register.”                      In the “External interrupt management” section, changed “This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.” to “The bits in the WISR[EIF] field are cleared by writing a ‘1’ to them; this prevents inadvertent overwriting of other flags in the register.”                      Revised the “WKPU block diagram” figure.</p> <p>In the WIFER section, deleted “The number of wakeups ... 1 and 18”.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>Revised the definitions of the following registers:</p> <ul style="list-style-type: none"> <li>– WISR</li> <li>– IRER</li> <li>– WRER</li> <li>– WIREER</li> <li>– WIFEER</li> <li>– WIFER</li> <li>– WIPUER</li> </ul> <p>In the Overview section, replaced the wakeup vector mapping information with a table.                      In the “WKPU memory map” table, added the module base address.                      In the NCR[NWRE0] field description, added a note about the proper sequence for enabling the NMI.</p> <p>Real Time Clock / Autonomous Periodic Interrupt                      Replaced ipg_clk with “system clock”.                      Changed “32 kHz” to “32 KHz”.                      Revised the RTCC[FRZEN] field description.                      In the “RTC functional description” section, revised the paragraph on clock sources.                      In the “RTC functional description” section, deleted “The RTCC[RTCVAL] field may only be updated when the RTCC[NTEN] bit is cleared to disable the counter”.                      In the “RTC/API register map” table, added the module base address.</p> <p>Chapter CAN Sampler                      Deleted the duplicate register map.                      In the “CAN Sampler memory map” table, added the module base address.                      In the “Internal multiplexer correspondence” table, revised the entries in the “Rx selected” column.</p> <p>Chapter e200z0h Core                      In the “e200z0h block diagram” figure, added a box around the core elements.</p> <p>Chapter Enhanced Direct Memory Access                      In the “DMA Clear Error (EDMA_CER)” section, corrected the offset from 0x001E to 0x001D.                      Deleted the “eDMA 32-bit memory map” table (information already present in the “eDMA memory map” table).                      In the “eDMA memory map” table, added the module base address.</p> <p>Chapter eDMA Channel Multiplexer                      Changed the chapter title (was “DMA Channel Multiplexer”, is “eDMA Channel Multiplexer”) and changed “DMA” to “eDMA” as appropriate to match the title.                      In the CHCONFIG register figure, revised the bit order (was 7...0, is 0...7) to match Power Architecture convention.                      In the “DMA_MUX memory map” table, added the module base address.</p> <p>Chapter Interrupt Controller                      Revised the INTC_IACKR section to illustrate the register’s dependence on INTC_MCR[VTES] more clearly.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>In the INTC_EOIR register figure, added “See text” to the W row.</p> <p>In the “Interrupt vector table” table:</p> <ul style="list-style-type: none"> <li>– Changed “WKUP” to “WKPU”.</li> <li>– Changed IRQs 63 and 83 to “Reserved”.</li> </ul> <p>Revised the “Interrupt sources available” table.</p> <p>In the “INTC memory map” table, added the module base address.</p> <p><b>Chapter Crossbar Switch</b> Renamed the chapter (was “Multi-Layer AHB Crossbar Switch”, is “Crossbar Switch”) and replaced the entire chapter contents.</p> <p><b>Chapter Memory Protection Unit</b> Removed extraneous content from the Introduction section. In the “MPU block diagram” figure, changed the text at the top left to “Platform” and removed “z0hn1 or”.</p> <p>Revised the MPU_CESR definition. Revised the MPU_EDRn definition. Revised the MPU_RGDn.Word2 definition. Revised the MPU_RGDAACn definition. Revised the Introduction section. Changed AHB to XBAR. Deleted references to IPS and replaced with “peripheral” as needed. In the “MPU access evaluation macro” figure, changed “AHB_ap” to “System bus address phase”.</p> <p>In the “MPU memory map” table, added the module base address.</p> <p><b>Chapter System Integration Unit Lite</b> Changed “WARNING” to “CAUTION”.</p> <p>In the register figures, changed “Access: None” to the corresponding actual level of access.</p> <p>Revised the description of the PARTNUM field in MIDR1 and MIDR2 to clarify that the field is split between the two registers.</p> <p>In the PCRx section, revised the WPS and WPE field descriptions to indicate the correct functionality.</p> <p>In the “External interrupts” section, changed “This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.” to “The bits in the ISR[EIF] field are cleared by writing a ‘1’ to them; this prevents inadvertent overwriting of other flags in the register.”</p> <p>In the “Peripheral input pin selection” table, changed the entry for PSMI48_51, PADSEL50, 01 (was PCR[48], is PCR[148]).</p> <p>Revised the “MIDR2 field descriptions” table to show how to calculate total flash memory size.</p> <p>In the “MIDR2 field descriptions” table, deleted the entry for FR (not implemented).</p> <p>In the “Peripheral input pin selection” table, added an entry for PADSEL62 and revised the footnotes accordingly.</p> <p>In the “SIUL memory map” table, added the module base address.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>In the IBCR section, changed “MS/SL” to “MSSL” and “Tx/Rx” to “TXRX” to ensure compliance with field name convention.</p> <p>In the IBSR figure, changed the IBAL and IBIF fields to w1c.</p> <p>In the IBCR figure, changed bit 7 (was IBDOZE, is reserved).</p> <p>In the “Interrupt description” section, changed “(TCF bit set - To be checked)” to “(a Byte Transfer interrupt occurs whenever the TCF bit changes from 0 to 1, that is, Transfer Under Progress to Transfer Complete state)”.</p> <p>Revised the last paragraph of the Overview section.</p> <p>In the IBCR[MDIS] field description, added “Status register bits (IBSR) are not valid when module is disabled”.</p> <p>In the IBSR[RXAK] field description, added “This bit is valid only after transfer is complete”.</p> <p>In the “Interrupt description” section, revised the entry for “Byte transfer condition”.</p> <p>In the “Initialization sequence” section, changed IBCR[IBDIS] to IBCR[MDIS].</p> <p>Revised the “Post-transfer software response” section.</p> <p>Added the “Transmit/receive sequence” section.</p> <p>In the “Generation of STOP” section, in the code sample, changed “bit 1” to “bit 5”.</p> <p>In the “I2C memory map” table, added the module base address.</p> <p><b>Chapter LIN Controller (LINFlex)</b></p> <p>In the “IFER field descriptions” table, switched “activated” and “deactivated” in order to match with “IFER[FACT] configuration” table.</p> <p>Deleted the “Register map and reset values” section (duplicate content).</p> <p>In the “UART mode” section, in the “9-bit frames” subsection, changed “sum of the 7 data bits” to “sum of the 8 data bits”.</p> <p>In the LINCR1[BF] field description, changed “this bit is reserved” to “this bit is reserved and always reads 1”.</p> <p>Changed “kbps” to “Kbit/s”.</p> <p><b>Chapter LIN Controller (LINFlexD)</b></p> <p>Changed layout of LINIBRR register figure to a single 32 bit format.</p> <p>In the register figures:</p> <ul style="list-style-type: none"> <li>– Added “Access: User read/write” to all register figures.</li> <li>– Updated instances of “These fields are writable only in Initialization mode.” to “These fields are writable only in Initialization mode (LINCR1[INIT] = 1).”.</li> </ul> <p>In the LINESR figure, changed the footnote “If LINTCSR[LTOM] is set, these fields are read-only.” to read “If LINTCSR[LTOM] = 1, these fields are read-only.”</p> <p>In the LINTOCR figure, added the footnote “The HTO field can only be written in slave mode, LINCR1[MME] = 1”.</p> <p>In the “Filter submodes” section, changed “eight IFCR registers” to “16 IFCRs” and “eight identifiers” to “16 identifiers”.</p> <p>In the “9-bit data frame” section, changed “The 8-bit UART data frame” to “The 9-bit UART data frame” and “sum of the 7 data bits” to “sum of the 8 data bits”.</p> <p>Revised the IFER section.</p> <p>In the LINCR1[MME] field description, changed “Master and Slave mode enable” to “Slave mode enable”.</p> <p>Revised the “Memory map and register description” section to show the differences in register availability on the various LINFlexD modules on this chip.</p> <p>In the LINCR1[BF] field description, changed “this bit is reserved” to “this bit is reserved and always reads 1”.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>Changed “kbps” to “Kbit/s”.</p> <p>In the “TCD chain memory map (master node, TX mode)” figure, changed the second instance of “Extended Frame (n+2)” to “Extended Frame (n+3)”.</p> <p>In the “TCD chain memory map (master node, RX mode)” figure, changed the second instance of “Extended Frame (n+1)” to “Extended Frame (n+2)”.</p> <p>In the “TCD chain memory map (slave node, TX mode)” figure, changed the second instance of “Extended Frame (n+1)” to “Extended Frame (n+2)”.</p> <p>In the “TCD chain memory map (slave node, RX mode)” figure, changed the second instance of “Extended Frame (n+1)” to “Extended Frame (n+2)”.</p> <p><b>Chapter FlexCAN</b>            Changed the chapter title (was “FlexCAN module”, is “FlexCAN”).            Deleted references to Stop mode (not supported on this chip).            In the CTRL field descriptions, added “0” and “1” to indicate what the bit values of 0 and 1 mean, respectively.            In the “Modes of operation” section, revised the description of Module Disable mode.            Revised the “Module Disable mode” section.            In the “FlexCAN memory map” table, added the module base addresses.</p> <p><b>Chapter Deserial Serial Peripheral Interface</b>            In the DSPIx_CTARn[LSBFE] field description, deleted the text “When operating in TSB configuration, this bit should be always 1.”.            In the “Continuous selection format” section, added a note about filling the TX FIFO.            Added new rules to the “Continuous serial communications clock” section.            Removed references to External Stop mode.            In the DSPI_SR[EOQF] field description, changed “The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming tidbit is sampled, but before the task delay starts.” to “The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached.”            In the DSPIx_CTARn[LSBFE] field description, deleted the text “When operating in TSB configuration, this bit should be always 1.”.            In the “DSPI memory map” table, added the module base addresses.</p> <p><b>Chapter Timers</b>            Added this chapter (incorporates content from STM, eMIOS, and PIT chapters).</p> <p><b>Chapter Analog-to-Digital Converter</b>            Updated MCR[WLSIDE] bit description.            Updated CDR register.            Replaced ADCDig with ADC, rewriting content as necessary.            In the PDEDR[PDED] field description, added “The delay is to allow time for the ADC power supply to settle before commencing conversions.”.            Replaced “ipg_clk” and “system clock” with “MC_PLL_CLK”.            In the THRHLR section, revised the addresses for THRHLR4–5 to align with the memory map.            Revised the CWSELR[0..11] section to show all combinations of registers for both ADCs.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>In the “12-bit ADC_1 digital registers” table, changed the address offsets for:</p> <ul style="list-style-type: none"> <li>– CIMR1</li> <li>– DMAR1</li> <li>– PSR1 (and adjacent reserved space)</li> <li>– CTR1</li> <li>– NCMR1</li> <li>– JCMR1 (and adjacent reserved space)</li> </ul> <p>Deleted the “Bit access descriptions” table.</p> <p>Chapter Cross Triggering Unit Removed remaining references to CTU_CSR (not implemented on this chip). In the “CTU memory map” table, changed the end address of the reserved space (was 0x002C, is 0x002F). In the “CTU memory map” table, added the module base address.</p> <p>Chapter Flash Memory Replaced the entire chapter.</p> <p>Chapter Static RAM In the Introduction section, replaced the text “Except in standby mode...” with the “SRAM behavior in chip modes” table.</p> <p>Chapter Register Protection Added this chapter.</p> <p>Chapter Software Watchdog Timer Added this chapter.</p> <p>Chapter Error Correction Status Module Revised the Introduction section. Revised the Features section. Revised the MUDCR section to show completely that bit 1 is reserved. In the register descriptions, revised the names as needed to match the names in the memory map. In the PREMR section, added text on where to find bus master IDs. Aligned register names in the descriptions and the memory map. Deleted the second paragraph in the Introduction section. Deleted the last bullet (about spp_ips_reg_protection) in the Features section. In the PREAT field descriptions, changed “AMBA-AHB” to “XBAR”. Renamed the “Spp_ips_reg_protection” section to “Register protection” and revised the section. Revised the “ECC registers” section. In the “ECSM memory map” table, added the module base address.</p> <p>Chapter IEEE 1149.1 Test Access Port Controller In the Features section, changed “3 test data registers” to “2 test data registers”. In the “SAMPLE instruction” section, added information about pad status. In the “SAMPLE/PRELOAD instruction” section, added information about pad status.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
19-Jul-2011	7 (cont.)	<p>Chapter Nexus Development Interface                      Added the “NPC_HNDSHK module” section.                      Changed STANDBY0 to STANDBY.                      Changed HALT0 to HALT.                      Changed STOP0 to STOP.</p> <p>Chapter Boot Assist Module                      Deleted this chapter (relevant content is now represented by the “Microcontroller Boot” chapter).</p> <p>Chapter Enhanced Modular IO Subsystem                      Deleted this chapter (relevant content is now represented by the “Timers” chapter).</p> <p>Chapter System Status and Configuration Module                      Deleted this chapter (relevant content is now represented by the “Microcontroller Boot” chapter).</p> <p>Appendix: Register Under Protection                      Deleted this appendix (relevant content is now represented by the “Register Protection” chapter).</p>
18-Sep-2013	8	Updated Disclaimer.
01-Jul-2015	9	<p><i>Chapter 1: Preface:</i>                      Editorial change                      In <a href="#">Table 1: Guide to this reference manual</a>, Line 12 WKPU changed external (pin) from 30 to 27                      In <a href="#">Table 1: Guide to this reference manual</a>, changed the description of Line 17 DMA_MUX.                      In <a href="#">Section 1.6.1: The SPC560B54/6x document set</a>, change bullet item “Configuring CPU memory, branch and cache optimizations” to “Configuring CPU memory and branch optimizations.”                      In <a href="#">Section 1.7.3: Software design</a>: remove the paragraph “The MMU translates physical memory addresses for use by the CPU and it must be configured before any peripherals or memories are available for use by the CPU. See the e200z4 Power Architecture Core Reference Manual for details on how to configure the MMU.”</p> <p><i>Chapter 2: Introduction:</i>                      In <a href="#">Table 3: SPC560B54/6x family comparison</a> updated the channel number in the row SCI (LINFlex): 8 channel for LQFP144 package and 10 channel for LQFP176 package.</p> <p><i>Chapter 4: Signal description</i>                      In <a href="#">Table 10: Functional port pin descriptions</a> removed Table Footnote 12.</p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
01-Jul-2015	9 (cont.)	<p><i>Chapter 6: Clock Description:</i>                      Add <b>Note:</b> to <a href="#">Section 6.8.4.1: Crystal clock monitor</a>.  <b>Note:</b> Functional FXOSC monitoring can only be guaranteed when the FXOSC frequency is greater than <math>(FIRC / 2RCDIV) + 0.5</math> MHz.                      Add <b>Note:</b> to <a href="#">Section 6.8.4.2: FMPLL clock monitor</a>.  <b>Note:</b> Functional FMPLL monitoring can only be guaranteed when the FMPLL frequency is greater than <math>(FIRC / 4) + 0.5</math> MHz.                      Updated <a href="#">Section 6.7.7: Recommendations</a></p> <p><i>Chapter 9: Reset Generation Module (MC_RGM):</i>                      Replaced <a href="#">Section 9.4.7: Boot mode capturing</a> with a new section.</p> <p><i>Chapter 10: Power Control Unit (MC_PCU):</i>                      Updated <a href="#">Section 10.1.2: Features</a>                      Updated <a href="#">Table 91: MC_PCU register description</a>.</p> <p><i>Chapter 11: Voltage Regulators and Power Supplies:</i>                      Updated <a href="#">Figure 95: Power domain organization</a>.</p> <p><i>Chapter 12: Wakeup Unit (WKPU):</i>                      Editorial change</p> <p><i>Chapter 13: Real Time Clock / Autonomous Periodic Interrupt (RTC/API):</i>                      In <a href="#">Table 109: RTCC field descriptions</a> update the Note in the RTCC[APIVAL] field description:</p> <p><i>Chapter 16: Enhanced Direct Memory Access (eDMA):</i>                      Updated <a href="#">Section 16.5.8: Dynamic programming</a>:                      added <a href="#">Section 16.5.8.1: Dynamic channel linking</a> and <a href="#">Section 16.5.8.2: Dynamic scatter/gather</a>                      Added <a href="#">Table 138: Coherency model for a dynamic channel link request</a>, <a href="#">Table 139: Coherency model for method 1</a> and <a href="#">Table 140: Coherency model for method 2</a>                      Added Method 1 (channel not using major loop channel linking) section and Method 2 (channel using major loop linking) section</p> <p><i>Chapter 18: Interrupt Controller (INTC):</i>                      Removed "User" access from <a href="#">Figure 149: INTC Module Configuration Register (INTC_MCR)</a>, <a href="#">Figure 150: INTC Current Priority Register (INTC_CPR)</a>, <a href="#">Figure 151: INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 0</a>, <a href="#">Figure 152: INTC Interrupt Acknowledge Register (INTC_IACKR) when INTC_MCR[VTES] = 1</a>, <a href="#">Figure 154: INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])</a>, <a href="#">Figure 155: INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])</a>, <a href="#">Figure 156: INTC Priority Select Register 0–3 (INTC_PSR[0:3])</a> and <a href="#">Figure 157: INTC Priority Select Register 232-233 (INTC_PSR[232:233])</a>.</p>



**Table 567. Document revision history (continued)**

Date	Revision	Changes
01-Jul-2015	9 (cont.)	<p><i>Chapter 19: Crossbar Switch (XBAR):</i>                      Throughout chapter correct “two master ports” to “three master ports” as necessary                      Updated <a href="#">Figure 160: XBAR block diagram</a>                      In <a href="#">Table 157: XBAR switch ports for SPC560B54/6x</a> add the row for eDMA                      In <a href="#">Section 19.4: Features</a> add a bullet item for eDMA.                      Updated <a href="#">Table 158: Hardwired bus master priorities</a></p> <p><i>Chapter 20: Memory Protection Unit (MPU):</i>                      In <a href="#">Figure 162: MPU Control/Error Status Register (MPU_CESR)</a>: expand the SPERR field to an 8-bit field stretching from bit 0 to bit 7.</p> <p><i>Chapter 22: Inter-Integrated Circuit Bus Controller Module (I2C):</i>                      Updated <a href="#">Section 22.6.1.5: Generation of STOP</a></p> <p><i>Chapter 23: LIN Controller (LINFlex):</i>                      Added Section Overrun after Section Error handling                      In <a href="#">Figure 225: Identifier filter control register (IFCR2n)</a> the ID field type should be changed from "w1c" to standard "read write" type                      In <a href="#">Figure 213: LIN output compare register (LINOOCR)</a>, update Footnote and replaced LINTCSR[LTOM] = 1 with LINTCSR[LTOM] = 0.                      In section “Identifier filter mode configuration” replaced “deactivated” with “activated” and IFER[FACT] = 0 with IFER[FACT] = 1.                      Beginning of <a href="#">Section 23.8.3.1: LIN timeout mode</a> replaced “Setting” with “Resetting”                      Beginning of <a href="#">Section 23.8.3.2: Output compare mode</a> replaced “Programming LINTCSR[LTOM] = 0” with “Programming LINTCSR[LTOM] = 1”.</p> <p><i>Chapter 24: LIN Controller (LINFlexD):</i>                      Added Section “Overrun” after Section “Error handling and detection”                      In <a href="#">Section 24.7.3.2: Identifier filter submode configuration</a> replace “deactivated” with “activated”.                      and replace “clearing” with “setting”                      In <a href="#">Figure 257: LIN output compare register (LINOOCR)</a>, update Footnote and replaced LINTCSR[LTOM] = 1 with LINTCSR[LTOM] = 0.                      Beginning of <a href="#">Section 24.12.1.1: LIN timeout mode</a> replaced “Setting” with “Resetting”                      Beginning of <a href="#">Section 24.12.1.2: Output compare mode</a> replaced “Resetting” with “Setting”.</p> <p><i>Chapter 25: FlexCAN:</i>                      Removed references throughout the chapter to “low cost MCUs”                      Removed Note above <a href="#">Table 288: FlexCAN memory map</a>                      Added Note in the RTR field description of <a href="#">Table 290: Message Buffer Structure field description</a></p>

**Table 567. Document revision history (continued)**

Date	Revision	Changes
01-Jul-2015	9 (cont.)	<p>Removed Note in <a href="#">Section 25.4.4.13: Rx Individual Mask Registers (RXIMR0–RXIMR63)</a></p> <p>Removed Note at end of <a href="#">Section 25.5.6: Matching process</a></p> <p>Updated Note following <a href="#">Figure 323: CAN Engine Clocking Scheme</a></p> <p>Updated title of <a href="#">Table 308: Bosch CAN 2.0B standard compliant bit time segment settings</a> from “CAN Standard Compliant Bit Time Segment Settings”</p> <p>In <a href="#">Section 25.5.9.4: Protocol Timing</a> updated note following <a href="#">Table 308: Bosch CAN 2.0B standard compliant bit time segment settings</a></p> <p>In <a href="#">Figure 312: Module Configuration Register (MCR)</a> updated WAK_MSK and WAK_SRC bit access.</p> <p>In <a href="#">Table 294: MCR field descriptions</a> removed row corresponding to WAK_MSK and WAK_SRC field description.</p> <p>Removed "For MCUs supporting individual masks per MB" in <a href="#">Section 25.4.4.4: Rx Global Mask (RXGMASK)</a>, in <a href="#">Section 25.4.4.5: Rx 14 Mask (RX14MASK)</a> and in <a href="#">Section 25.4.4.6: Rx 15 Mask (RX15MASK)</a>.</p> <p>In <a href="#">Table 296: CTRL field descriptions</a>, in CLK_SRC field removed Note.</p> <p>Updated <a href="#">Section 25.5.1: Overview</a>.</p> <p><b>Chapter 27: Timers:</b></p> <p>Added Note in <a href="#">Section 27.4.4.2: Output Pulse Width and Frequency Modulation Buffered (OPWFMB) mode</a>.</p> <p><b>Chapter 28: Analog-to-Digital Converter (ADC):</b></p> <p>In <a href="#">Section 28.3.4.2: CTU in trigger mode</a> replace the sentence If another CTU conversion is triggered before the end of the conversion, that request is discarded. with:</p> <p>If another CTU conversion is triggered before the end of the conversion, that request is discarded. However, if the CTU has triggered a conversion that is still ongoing on a channel, it will buffer a second request for the channel and wait for the end of the first conversion before requesting another conversion. Thus, two conversion requests close together will both be serviced.</p> <p>In <a href="#">Section 28.3.5.2: Presampling channel enable signals</a> in <a href="#">Table 379: Presampling voltage selection based on PREVALx fields</a> in the 01 row, change the “Presampling voltage” field to: V1 = VDD_HV_ADC0 or VDD_HV_ADC1.</p> <p>Add Note to <a href="#">Section 28.3.11: Auto-clock-off mode</a></p> <p>In <a href="#">Table 384: MSR field descriptions</a> updated description of ADCSTATUS field.</p> <p><b>Chapter 29: Cross Triggering Unit (CTU):</b></p> <p>Add a Note at the end of <a href="#">Section 29.4.1: Event Configuration Registers (CTU_EVTCFGRx) (x = 0..63)</a></p>

Table 567. Document revision history (continued)

Date	Revision	Changes
01-Jul-2015	9 (cont.)	<p><i>Chapter 30: Flash Memory:</i> Updated <a href="#">Figure 488: Flash memory architecture</a> Added Note below <a href="#">Section 30.7.2: Memory map and register description</a>. Updated <a href="#">Section 30.8.6: Access pipelining</a></p> <p><i>Chapter 31: Static RAM (SRAM):</i> In <a href="#">Table 503: Low power configuration</a> in the STANDBY line, change the description "Either all or just 8 KB of the SRAM remains powered. This option is software-selectable." to "Either 32 KB or just 8 KB of the SRAM remains powered. This option is software-selectable."</p> <p><i>Chapter 32: Register Protection:</i> In <a href="#">Table 510: Protected registers</a> change the module base address for the CMU_CSR register from C3FE00E0 to C3FE0000.</p> <p><i>Chapter 36: Nexus Development Interface (NDI):</i> Editorial change</p>
25-Jan-2016	10	<p><i>Chapter 2: Introduction:</i> in <a href="#">Table 3: SPC560B54/6x family comparison</a>, added column relating to "LQFP100" package in SPC560B64 devices.</p>

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved