

UG0331
User Guide
SmartFusion2 Microcontroller Subsystem



Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

© 2018 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

Contents

1	Revision History	1
1.1	Revision 15.0	1
1.2	Revision 14.0	1
1.3	Revision 13.0	1
1.4	Revision 12.0	1
1.5	Revision 11.0	2
1.6	Revision 10.0	2
1.7	Revision 9.0	2
1.8	Revision 8.0	2
1.9	Revision 7.0	2
1.10	Revision 6.0	3
1.11	Revision 5.0	3
1.12	Revision 4.0	3
1.13	Revision 3.0	4
1.14	Revision 2.0	4
1.15	Revision 1.0	4
2	Cortex-M3 Processor Overview and Debug Features	6
2.1	Features	6
2.2	Functional Description	7
2.3	Cortex-M3 Processor NVIC	8
2.4	Cortex-M3 Processor SysTick Timer	12
2.5	Cortex-M3 Processor Debug Subsystem	12
2.5.1	Cortex-M3 Processor Debug Port	12
2.5.2	Cortex-M3 Processor Trace System	13
2.6	Cortex-M3 Processor Port Descriptions	15
2.7	How to Use the Cortex-M3 Processor and the Debug Subsystem	16
2.7.1	Configuration Through Libero Software and Firmware	16
3	Cortex-M3 Processor (Reference Material)	18
3.1	System Level Interface	19
3.2	Integrated Configurable Debug	19
3.3	Cortex-M3 Processor Features and Benefits Summary	19
3.4	Cortex-M3 Processor Core Peripherals	20
3.4.1	Nested Vectored Interrupt Controller	20
3.4.2	System Control Block	20
3.4.3	System Timer	20
3.4.4	Memory Protection Unit	20
3.5	Cortex-M3 Processor Description	20
3.5.1	Programmers Model	20
3.5.2	Memory Model	29
3.5.3	Exception Model	37
3.5.4	Fault Handling	43
3.5.5	Power Management	45
3.6	Cortex-M3 Processor Instruction Set	47
3.6.1	Instruction Set Summary	47
3.6.2	CMSIS Functions	50
3.6.3	About the Instruction Descriptions	51

3.6.4	Memory Access Instructions	58
3.6.5	General Data processing instructions	68
3.6.6	Multiply and Divide Instructions	77
3.6.7	Saturating Instructions	79
3.6.8	Bitfield instructions	81
3.6.9	Branch and Control Instructions	83
3.6.10	Miscellaneous Instructions	89
3.7	Cortex-M3 Processor Peripherals	95
3.7.1	About the Cortex-M3 Processor Peripherals	95
3.7.2	System Control Block	102
3.7.3	System Timer, SysTick	119
3.7.4	Memory Protection Unit	122
4	Cache Controller	133
4.1	Features	133
4.2	Functional Description	134
4.2.1	Cache Matrix	134
4.2.2	Memory Mapping	134
4.2.3	Memory Maps and Transaction Mapping	136
4.2.4	Cache Locked Mode	141
4.2.5	Interfaces	142
4.3	How to Use Cache Controller	143
4.3.1	System Registers Used for Cache Operations	144
5	Embedded NVM (eNVM) Controllers	145
5.1	Features	145
5.2	Functional Description	146
5.2.1	Memory Organization	147
5.2.2	Data Retention Time	148
5.2.3	eNVM Access Time	148
5.2.4	Theory of Operation	148
5.2.5	eNVM Command Register	151
5.2.6	Error Response	159
5.2.7	Interrupt to Cortex-M3 Processor	159
5.3	Security	159
5.3.1	User Protectable 4K Regions	160
5.3.2	eNVM Pages for Special Purpose Storage	163
5.4	How to Use eNVM	165
5.4.1	Data Storage in eNVM Using the Libero eNVM Client	165
5.4.2	Reading the eNVM Block	172
5.4.3	Writing to the eNVM Block	173
5.4.4	Firmware and Sample Project	173
5.5	SYSREG Control Registers	174
5.6	eNVM Control Registers	180
5.6.1	Status Register Bit Definitions	184
6	Embedded SRAM (eSRAM) Controllers	187
6.1	Features	187
6.2	Functional Description	188
6.2.1	Memory Organization	190
6.2.2	Modes of Operation	191
6.2.3	Pipeline Modes and Wait States for Read and Write Operations	191
6.3	How to Use eSRAM	194
6.4	SYSREG Control Registers	198
7	AHB Bus Matrix	210

7.1	Functional Description	211
7.1.1	Architecture Overview	211
7.1.2	Timing Diagrams	214
7.1.3	Details of Operation	218
7.1.4	System Memory Map	225
7.2	How to Use AHB Bus Matrix	233
7.2.1	Design Flow	233
7.3	Register Map	235
8	High Performance DMA Controller	236
8.1	Features	237
8.2	Functional Description	237
8.2.1	Initialization	240
8.2.2	Details of Operation	241
8.3	How to Use HPDMA	241
8.3.1	Design Flow	241
8.3.2	HPDMA Use Models	245
8.4	HPDMA Controller Register Map	246
8.4.1	HPDMA Register Bit Definitions	247
8.5	SYSREG Control Register	263
9	Peripheral DMA	264
9.1	Features	264
9.2	Functional Description	265
9.2.1	Architecture Overview	265
9.2.2	Port List	268
9.2.3	Initialization	269
9.2.4	Details of Operations	269
9.3	How to Use the PDMA	271
9.3.1	Design Flow	271
9.3.2	PDMA Use Models	274
9.4	PDMA Register Map	275
9.4.1	PDMA Configuration Register Bit Definitions	278
9.5	SYSREG Control Registers	283
10	Universal Serial Bus OTG Controller	284
10.1	Features	284
10.2	Functional Description	285
10.2.1	Architecture Overview	285
10.2.2	USB OTG Controller Interface Signals	287
10.2.3	USB OTG Controller Operations	291
10.3	How to Use USB OTG Controller	300
10.3.1	Libero Settings for USB OTG Configuration	300
10.3.2	Software: Firmware, USB Class Specific Code, and Application Code	303
10.3.3	USB OTG Controller Clocks and Resets	308
10.3.4	Programmability	308
10.3.5	Common Registers	309
10.3.6	Indexed Registers	315
10.3.7	FIFO Registers	330
10.3.8	Control and Status Registers (OTG, Dynamic FIFO and Version)	331
10.3.9	ULPI and Configuration Registers	338
10.3.10	Non-Indexed End Point Control/Status Registers	344
10.3.11	Extended Registers	351
10.3.12	Direct Memory Access (DMA) Registers	354
10.3.13	Multipoint Control and Status Registers	358
10.3.14	Link Power Management Registers	366

10.3.15	USB System Registers	371
11	Ethernet MAC	374
11.1	Features	374
11.2	Functional Description	375
11.2.1	EMAC Functional Blocks	375
11.3	TSEMAC PHY Interfaces	377
11.4	EMAC Operation	381
11.4.1	Transmit Operation	382
11.4.2	Receive Operation	383
11.5	How to Use TSEMAC	384
11.5.1	SGMII Interface Configuration	387
11.5.2	SECEDED Features for TSEMAC Buffers	389
11.6	SYSREG Control Register for EMAC	392
11.7	EMAC Configuration Register Summary	393
11.8	EMAC Register Bit Definitions	399
11.9	CoreMACFilter Overview	434
11.9.1	Features	434
12	CAN Controller	436
12.1	Features	436
12.1.1	Compliance	436
12.1.2	Receive Path	436
12.1.3	Transmit Path	437
12.1.4	EDAC	437
12.1.5	Enable or Disable Control	437
12.1.6	System Dependencies	437
12.2	Functional Description	437
12.2.1	CAN Controller Interface Signals	437
12.2.2	Transmit Procedures	438
12.2.3	Receive Procedures	439
12.2.4	Interrupt Generation	441
12.2.5	CAN Test Modes	442
12.3	CAN Controller Configuration	442
12.3.1	Peripheral Signals Assignment Table	443
12.3.2	EDAC CAN Configuration	444
12.4	How to Use the MSS CAN Controller	446
12.4.1	Hardware Design Flow	446
12.5	Use Cases	448
12.5.1	Use Case 1: Automatic Bit Rate Detection	448
12.5.2	Use Case 2: SRAM Test Mode	448
12.6	CAN Controller Register Map	451
12.6.1	SYSREG Control Registers	451
12.6.2	CAN Controller Registers	452
12.6.3	Configuration Register	454
12.6.4	Command Register	455
12.6.5	Transmit Message Control and Command Register	456
12.6.6	Transmit Buffer Status Register	459
12.6.7	Receive Message Control and Command Register	460
12.6.8	Receive Buffer Status Register	464
12.6.9	Error Capture Register	465
12.6.10	Error Status Register	466
12.6.11	Interrupt Registers	466
13	MMUART Peripherals	469

13.1	Features	469
13.2	Functional Description	470
13.2.1	Architecture Overview	470
13.2.2	Port List	471
13.2.3	Initialization	473
13.2.4	Details of Operation	474
13.3	How to Use MMUART	486
13.3.1	Design Flow	486
13.3.2	MMUART Use Models	489
13.4	MMUART Register Map	490
13.4.1	Receiver Buffer Register (RBR)	491
13.4.2	Transmit Holding Register (THR)	491
13.4.3	FIFO Control Register (FCR)	491
13.4.4	Baud Rate Divisor Registers	492
13.4.5	Interrupt Enable Register (IER)	494
13.4.6	Multi-Mode Interrupt Enable Register (IEM)	494
13.4.7	Interrupt Identification Register (IIR)	495
13.4.8	Multi-Mode Interrupt Identification Register (IIM)	496
13.4.9	Line Control Register (LCR)	496
13.4.10	Modem Control Register (MCR)	497
13.4.11	Line Status Register (LSR)	498
13.4.12	Modem Status Register (MSR)	499
13.4.13	Scratch Register (SR)	500
13.4.14	Multi-Mode Control Register 0 (MM0)	500
13.4.15	Multi-Mode Control Register 1 (MM1)	501
13.4.16	Multi-Mode Control Register 2 (MM2)	501
13.4.17	Glitch Filter Register (GFR)	502
13.4.18	Transmitter Time Guard Register (TTG)	503
13.4.19	Receiver Timeout Register (RTO)	503
13.4.20	Address Register (ADR)	503
14	Serial Peripheral Interface Controller	504
14.1	Features	504
14.2	Functional Description	505
14.2.1	Architecture Overview	505
14.2.2	Interface	506
14.2.3	Initialization	516
14.2.4	Details of Operation	517
14.3	How to Use SPI	519
14.3.1	Design Flow	520
14.3.2	SPI Use Models	525
14.4	SPI Register Map	527
14.4.1	SYSREG Configuration Register Summary	527
14.4.2	SPI Register Summary	527
14.4.3	SPI Register Details	528
15	Inter-Integrated Circuit Peripherals	538
15.1	Features	538
15.2	Functional Description	539
15.2.1	Architecture Overview	539
15.2.2	Port List	540
15.2.3	Initialization	541
15.2.4	Details of Operation	542
15.3	How to Use I ² C	544
15.3.1	Design Flow	544
15.3.2	I ² C Use Models	548

15.4	I ² C Register Map	550
15.4.1	Control Register	551
15.4.2	Status Register	552
15.4.3	Data Register	558
15.4.4	Slave0 Address Register	559
15.4.5	SMBus Register	559
15.4.6	Frequency Register	560
15.4.7	Glitch Register	561
15.4.8	Slave1 Address Register	561
16	MSS GPIO	562
16.1	Features	562
16.2	MSS GPIO Functional Description	563
16.2.1	MSS GPIO Configuration Registers (GPIO_X_CFG)	565
16.2.2	MSS GPIO Reset Functionality	566
16.3	MSS GPIO Usage	568
16.3.1	Configuring MSS GPIO Using Libero SoC	568
16.3.2	MSS GPIO Use Models	573
16.4	GPIO Register Map	574
16.4.1	SYSREG Block Registers	576
16.4.2	Software Reset Control Register	576
16.4.3	MSS GPIO Definitions	577
16.4.4	Loopback Control Register	577
16.4.5	GPIO Input Source Select Control Register	577
16.4.6	GPIO System Reset Control Register	578
16.4.7	I/O MUX Associated With GPIOs	578
17	Communication Block	592
17.1	Features	592
17.2	Functional Description	593
17.2.1	Architecture Overview	593
17.2.2	Frame/Command Marker	594
17.2.3	Clocks	595
17.2.4	Resets	595
17.2.5	Interrupts	595
17.2.6	COMM_BLK Initialization	595
17.2.7	CoreSysServices Soft IP	595
17.3	How to Use the Communication Block	596
17.3.1	COMM_BLK Configuration	596
17.3.2	Use Model	597
17.4	COMM_BLK Configuration Registers	597
17.5	COMM_BLK Register Interface Details	598
17.5.1	Control Register	598
17.5.2	Status Register	598
17.5.3	Interrupt Enable Register	599
17.5.4	Byte Data Register	599
17.5.5	Word Data Register	600
17.5.6	Frame/Command Byte Register	600
17.5.7	Frame/Command Word Register	600
18	RTC System	601
18.1	Features	601
18.2	Functional Description	602
18.2.1	Architecture Overview	602
18.2.2	Port List	603
18.2.3	Details of Operation	604

18.3	How to Use RTC	605
18.3.1	Design Flow	605
18.3.2	RTC Use Model	608
18.4	RTC Register Map	608
18.4.1	Counter Bit Positions	609
18.4.2	Register Bit Allocation	609
18.4.3	Control Register	610
18.4.4	Mode Register	611
18.4.5	Prescaler	611
18.4.6	Alarm and Compare Registers	612
18.4.7	Date and Time Registers	612
18.5	SYSREG Control Registers	613
19	System Timer	614
19.1	Features	614
19.2	Functional Description	614
19.2.1	Architecture Overview	614
19.2.2	Port List	615
19.2.3	Details of Operation	616
19.3	How to Use Timer	619
19.3.1	Design Flow	619
19.3.2	Timer Use Models	621
19.4	Timer Register Map	622
19.4.1	Timer x Value Register	623
19.4.2	Timer x Load Value Register	623
19.4.3	Timer x Background Load Value Register	623
19.4.4	Timer x Raw Interrupt Status Register	624
19.4.5	Timer x Masked Interrupt Status Register	625
19.4.6	Timer 64 Value Upper Register	625
19.4.7	Timer 64 Value Lower Register	625
19.4.8	Timer 64 Load Value Upper Register	625
19.4.9	Timer 64 Load Value Lower Register	626
19.4.10	Timer 64 Background Load Value Upper Register	626
19.4.11	Timer 64 Background Load Value Lower Register	627
19.4.12	Timer 64 Control Register	627
19.4.13	Timer 64 Raw Interrupt Status Register	628
19.4.14	Timer 64 Masked Interrupt Status Register	628
19.4.15	Timer 64 Mode Register	628
20	Watchdog Timer	629
20.1	Features	629
20.2	Functional Description	630
20.2.1	Architecture Overview	630
20.2.2	Port List	631
20.2.3	Details of Operation	631
20.3	How to Use the Watchdog Timer	633
20.3.1	Design Flow	634
20.3.2	Watchdog Timer Use Models	637
20.4	Watchdog Timer Register Map	638
20.4.1	Watchdog Timer Configuration Register Bit Definitions	638
20.5	SYSREG Control Registers	641
21	Reset Controller	642
21.1	Functional Description	643
21.1.1	Power-On Reset Generation Sequence	643
21.1.2	Power-Up to Functional Time Sequence	645

21.2	Power-Up to Functional Time Data	646
21.2.1	Parameters Used for Obtaining Power-Up to Functional Time Data	646
21.2.2	VDD Power-Up to Functional Time	647
21.2.3	DEVRST_N Power-Up to Functional Time	651
21.2.4	Power-On Reset	654
21.2.5	System Reset	654
21.2.6	Block Resets	656
21.3	CoreResetP Soft Reset Controller	660
21.3.1	Reset Topology	661
21.3.2	Implementation	663
21.3.3	Timing Diagrams	664
21.4	How to Use the Reset Controller	666
21.4.1	Ramp Delay Configuration	666
21.4.2	Reset Controller Configurator	666
21.5	SYSREG Control Registers	669
22	System Register Block	670
22.0.1	SYSREG Block Register Write Protection	670
22.0.2	Register Types	671
22.1	Register Lock Bits Configuration	674
22.1.1	Lock Bit File	675
22.1.2	Lock Bit File Syntax	675
22.1.3	Locking and Unlocking a Register	676
22.2	Register Map	676
22.3	Register Details	682
22.3.1	System Registers Behavior for M2S005/010 Devices	682
22.3.2	eSRAM Configuration Register	683
22.3.3	eSRAM Latency Configuration Register	683
22.3.4	DDR Configuration Register	684
22.3.5	eNVM Configuration Register	684
22.3.6	eNVM Remap Base Address Control Register	686
22.3.7	eNVM FPGA Fabric Remap Base Address Register	687
22.3.8	Cache Configuration Register	687
22.3.9	Cache Region Control Register	688
22.3.10	Cache Lock Base Address Control Register	688
22.3.11	Cache Flush Index Control Register	688
22.3.12	MSS DDR Bridge Buffer Timer Control Register	689
22.3.13	MSS DDR Bridge Non-Bufferable Address Control Register	689
22.3.14	MSS DDR Bridge Non-Bufferable Size Control Register	689
22.3.15	MSS DDR Bridge Configuration Register	690
22.3.16	EDAC Configuration Register	691
22.3.17	Master Weight Configuration Register 0	691
22.3.18	Master Weight Configuration Register 1	692
22.3.19	Software Interrupt Register	693
22.3.20	Software Reset Control Register	693
22.3.21	M3 Configuration Register	695
22.3.22	Fabric Interface Control (FIC) Register	695
22.3.23	Loopback Control Register	696
22.3.24	GPIO System Reset Control Register	696
22.3.25	GPIO Input Source Select Control Register	697
22.3.26	MDDR Configuration Register	697
22.3.27	USB I/O Input Select Control Register	698
22.3.28	Peripheral Clock MUX Select Control Register	698
22.3.29	Watchdog Configuration Register	699
22.3.30	MDDR I/O Calibration Control Register	699
22.3.31	EDAC Interrupt Enable Control Register	699
22.3.32	USB Configuration Register	701

22.3.33	eSRAM PIPELINE Configuration Register	701
22.3.34	RTC Wake Up Configuration Register	702
22.3.35	MAC Configuration Register	702
22.3.36	MSS DDR PLL Status Low Configuration Register	703
22.3.37	MSS DDR PLL Status High Configuration Register	704
22.3.38	MSS DDR Fabric Alignment Clock Controller (FACC) Configuration Register 1	705
22.3.39	MSS DDR Fabric Alignment Clock Controller Configuration Register 2	707
22.3.40	PLL LOCK Enable Control Register	709
22.3.41	MSS DDR Clock Calibration Control Register	709
22.3.42	PLL Delay Line Select Control Register	709
22.3.43	MAC Status Clear on Read Control Register	710
22.3.44	Reset Source Control Register	710
22.3.45	Dcode Bus Error Address Status Register	711
22.3.46	ICode Bus Error Address Status Register	711
22.3.47	System Bus Error Address Status Register	711
22.3.48	ICode Miss Control Status Register	711
22.3.49	ICode Hit Control Status Register	711
22.3.50	DCode Miss Control Status Register	712
22.3.51	DCode Hit Control Status Register	712
22.3.52	ICode Transaction count Control Status Register	712
22.3.53	DCode Transaction Count Control Status Register	712
22.3.54	MSS DDR Bridge DS master Error Address Status Register	713
22.3.55	MSS DDR Bridge High Performance DMA Master Error Address Status Register	713
22.3.56	MSS DDR Bridge AHB Bus Error Address Status Register	713
22.3.57	MSS DDR Bridge Buffer Empty Status Register	714
22.3.58	MSS DDR Bridge Disable Buffer Status Register	714
22.3.59	eSRAM0 EDAC Count	715
22.3.60	eSRAM1 EDAC Count	715
22.3.61	MAC EDAC Transmitter Count	715
22.3.62	MAC EDAC Receiver Count	715
22.3.63	USB EDAC Count	716
22.3.64	CAN EDAC Count	716
22.3.65	eSRAM0 EDAC Address Register	716
22.3.66	eSRAM1 EDAC Address Register	716
22.3.67	MAC EDAC Receiver Address Register	717
22.3.68	MAC EDAC Transmitter Address Register	717
22.3.69	CAN EDAC Address Register	717
22.3.70	USB EDAC Address Register	717
22.3.71	Security Configuration Register for Masters 0, 1, and 2	718
22.3.72	Security Configuration Register for Masters 4, 5, and DDR_FIC	718
22.3.73	Security Configuration Register for Masters 3, 6, 7, and 8	719
22.3.74	Security Configuration Register for Master 9	720
22.3.75	M3 Status Register	721
22.3.76	ETM Count Low Register	721
22.3.77	ETM Count High Register	721
22.3.78	Device Status Register	722
22.3.79	eNVM Protect User Register	722
22.3.80	Smart Fusion2 eNVM Status Register	724
22.3.81	Device Version Register	724
22.3.82	MSS DDR PLL Status Register	724
22.3.83	USB Status Register	725
22.3.84	eNVM Status Register	725
22.3.85	DDR8 Status Register	726
22.3.86	MDDR IO Calibration Status Register	726
22.3.87	MSS DDR Clock Calibration Status	727
22.3.88	Watch Dog Load Register	727
22.3.89	Watch Dog MVRP Register	727
22.3.90	User Configuration Register 0	727
22.3.91	User Configuration Register 1	727

22.3.92	User Configuration Register 2	728
22.3.93	User Configuration Register 3	728
22.3.94	Fabric Protected Size Register	728
22.3.95	Fabric Protected Base Address Register	729
22.3.96	MSS GPIO Definitions	730
22.3.97	EDAC Status Register	730
22.3.98	MSS Internal Status Register	731
22.3.99	MSS External Status Register	731
22.3.100	Watchdog Timeout Event	733
22.3.101	Clear MSS Counters	733
22.3.102	Clear EDAC Counters	733
22.3.103	Flush Configuration Register	734
22.3.104	MAC Status Clear Control Register	735
22.3.105	IOMUXCELL_CONFIG[n] Configuration Register	736
23	Fabric Interface Interrupt Controller	738
23.1	Features	738
23.2	Functional Description	739
23.2.1	Architecture Overview	739
23.2.2	FIIC Port List	741
23.3	How to Use FIIC	741
23.3.1	Configuring the FIIC Using the Libero SoC	741
23.3.2	FIIC Use Models	743
23.4	FIIC Controller Registers	749
23.5	FIIC Controller Register Bit Definitions	749
24	Fabric Interface Controller	757
24.1	Functional Description	758
24.1.1	MSS to the FPGA Fabric Interface	759
24.1.2	Configure FIC for Master or Slave Interface	759
24.2	Advanced AHB-Lite Options	759
24.2.1	Configure FIC in Bypass Mode or Synchronous Pipelined Mode	759
24.2.2	Master Identity Port to the Fabric	760
24.2.3	Configure MSS Master View for the FPGA Fabric Address	760
24.3	FIC Interface Port List	761
24.4	Timing Diagrams	763
24.5	Implementation Considerations	766
24.6	Fabric Interface Clocks	766
24.7	How to Use FIC	767
24.7.1	FIC Configuration	767
24.7.2	Configuring the FIC Subsystem Clocks	774
24.7.3	Configuring the FIC Subsystem Reset	778
24.7.4	Use Models	779
24.8	Reference Documents	782
24.9	SYSREG Control Registers for FIC_0 and FIC_1	783
25	APB Configuration Interface	784
25.1	Functional Block Diagram Description	784
25.1.1	Architecture Overview	784
25.1.2	Port List	785
25.1.3	CoreSF2Config Soft IP	786
25.2	How to Use	787
25.2.1	Configuring FIC_2 (Peripheral Initialization) Using Libero SoC	787
25.2.2	FIC_2 Use Models	790

26	Error Detection and Correction Controllers	792
26.1	Functional Description	792
26.1.1	EDAC Checksum Bits Width	793
26.2	Configuration	793
26.3	How to Use EDAC	794

Figures

Figure 1	Cortex-M3 Processor R2P1 Block Diagram as Implemented in the SmartFusion2 SoC FPGA	7
Figure 2	Trace System Block Diagram	14
Figure 3	CM3 Configurator	16
Figure 4	Cortex-M3 Processor Implementation	18
Figure 5	Core Register Set	21
Figure 6	Program Status Register	22
Figure 7	Priority Mask Register	25
Figure 8	Fault Mask Register	26
Figure 9	Base Priority Mask Register	26
Figure 10	Control Register	27
Figure 11	Processor Memory Map	29
Figure 12	Memory Ordering Restrictions	30
Figure 13	Bit-band Mapping	34
Figure 14	Byte-Invariant Big-Endian Format	35
Figure 15	Little Endian Format	35
Figure 16	Vector Table	40
Figure 17	Exception Entry Stack Contents	42
Figure 18	ASR#3	53
Figure 19	LSR	53
Figure 20	LSL	54
Figure 21	ROR	54
Figure 22	RRX	55
Figure 23	ISER Register Bit Assignments	96
Figure 24	ICER Register Bit Assignments	97
Figure 25	ISPR Register Bit Assignments	97
Figure 26	ICPR Register Bit Assignments	98
Figure 27	IABR Register Bit Assignments	98
Figure 28	IPR Register Bit Assignments	99
Figure 29	IABR Register Bit Assignments	100
Figure 30	ACTLR Bit Assignments	103
Figure 31	CPUID Register Bit Assignments	103
Figure 32	ICSR Bit Assignments	104
Figure 33	VTOR Bit Assignments	106
Figure 34	AIRCR Bit Assignments	107
Figure 35	SCR Bit Assignments	108
Figure 36	CCR Bit Assignments	109
Figure 37	SHPR1 Bit Assignments	111
Figure 38	SHPR2 Bit Assignments	111
Figure 39	SHPR3 Bit Assignments	111
Figure 40	SHCSR Bit Assignments	112
Figure 41	CFSR Bit Assignments	113
Figure 42	MMFSR Bit Assignments	114
Figure 43	MMFSR Bit Assignments	115
Figure 44	UFSR Bit Assignments	116
Figure 45	HFSR Bit Assignments	117
Figure 46	SYST_CTRL Register Bit Assignments	120
Figure 47	SYST_RVR Register Bit Assignments	120
Figure 48	SYST_CVR Register Bit Assignments	121
Figure 49	SYST_CALIB Register Bit Assignments	121
Figure 50	MPU_TYPE Register Bit Assignments	124
Figure 51	MPU_CTRL Register Bit Assignments	124
Figure 52	SYST_CVR Register Bit Assignments	125
Figure 53	MPU_RBAR Bit Assignments	126
Figure 54	MPU_RASR Bit Assignments	127

Figure 55	SRD Field	131
Figure 56	Cache Controller Interfaces to Cortex-M3 Processor, AHB Bus Matrix, and MDDR Bridge	133
Figure 57	Cache Controller Block Diagram	134
Figure 58	General Cache Architecture and Addressing	139
Figure 59	IAR Compiler Options	140
Figure 60	IAR Assembler Options	141
Figure 61	Cache Controller Interface	142
Figure 62	MSS Configurator with Cache Controller Configuration Options	143
Figure 63	MSS Configurator with Remapping Options for eNVM, eSRAM, and MDDR	143
Figure 64	System Builder with Remapping Options for eNVM, eSRAM, and MDDR	144
Figure 65	eNVM Connection to AHB Bus Matrix	145
Figure 66	eNVM Controller Block Diagram	146
Figure 67	Write Path	149
Figure 68	Read Path	150
Figure 69	Timing Diagram Showing Single Word Read Operation	154
Figure 70	Timing Diagram Showing Consecutive Reads Incrementing through Memory	154
Figure 71	Timing Diagram Showing Cache Fill Read Operations Utilizing Bursts	155
Figure 72	eNVM Program (ProgramADS) and Verify (VerifyADS) Operations	156
Figure 73	Exclusive Register Access and Filling Data in WDBUFF	156
Figure 74	Issuing the ProgramADS Command	156
Figure 75	Completion of ProgramADS and Issue of VerifyADS Command	157
Figure 76	Completion of eNVM Verify Operation	157
Figure 77	Complete eNVM Program and Verify Operations Waveform	158
Figure 78	Exclusive Register Access and Filling Data in WDBUFF	158
Figure 79	ProgramAD Command	158
Figure 80	ProgramDA Command	158
Figure 81	ProgramStart Command	159
Figure 82	eNVM Special Sectors for the M2S050TS Device with 256 KB eNVM_0	160
Figure 83	eNVM Special Sectors for the M2S005S Device with 128 KB eNVM_0	160
Figure 84	eNVM Special Sectors for the M2S010TS, M2S025TS Devices with 256 KB eNVM_0	161
Figure 85	eNVM Special Sectors for the M2S060TS Devices with 256 KB eNVM_0	161
Figure 86	eNVM Special Sectors for the M2S090TS, M2S150TS Devices with 512 KB	162
Figure 87	System Builder Window	166
Figure 88	System Builder - Device Features Tab	167
Figure 89	System Builder - Memories Tab	168
Figure 90	Add Data Storage Client Dialog	169
Figure 91	System Builder - Memories Tab with Two eNVM Clients	170
Figure 92	System Builder - Microcontroller Tab	171
Figure 93	System Builder - Security Tab	172
Figure 94	Firmware Catalog Showing the Generation of Sample Project for eNVM	173
Figure 95	eSRAM_0 and eSRAM_1 Connection to AHB Bus Matrix	188
Figure 96	eSRAM Controller Block Diagram	189
Figure 97	System Builder Window	194
Figure 98	System Builder - Microcontroller Tab	195
Figure 99	System Builder - SECDED Tab	196
Figure 100	System Builder - Security Tab	197
Figure 101	AHB Bus Matrix Masters and Slaves	210
Figure 102	Master Stage and Slave Stage Interconnection	212
Figure 103	Block Diagram of APB Destinations Connected to AHB Bus Matrix	213
Figure 104	AHB-Lite Write Transactions	214
Figure 105	AHB-Lite Read Transactions	215
Figure 106	AHB-to-AHB Write Transactions	216
Figure 107	AHB-to-AHB Read Transactions	217
Figure 108	Pure Round Robin and Fixed Priority Slave Arbitration Scheme	219
Figure 109	WRR and Fixed Priority Slave Arbitration Scheme	221
Figure 110	Slave Arbitration Flow Diagram	223
Figure 111	AHB Bus Matrix to Fabric Interface Controller	224
Figure 112	Default System Memory Map	225
Figure 113	Memory Map after eSRAM Remap (64 KB eSRAM)	226

Figure 114	Use Case for eSRAM Execution	227
Figure 115	Virtual eNVM View (After Chip Boot)	228
Figure 116	Virtual eNVM View for Soft Processor	229
Figure 117	DDR Memory Remap	230
Figure 118	AHB Bus Matrix in Libero SoC Design MSS Configurator	233
Figure 119	AHB Bus Matrix Configuration Window	234
Figure 120	HPDMA Interfacing With MSSDDR Bridge and AHB Bus Matrix	236
Figure 121	HPDMA Controller Block Diagram	237
Figure 122	HPDMA Registers	238
Figure 123	DMA Controller Flow Chart	239
Figure 124	Enable HPDMA in the Libero SOC Design MSS Configurator	242
Figure 125	HPDMA Transfers Data Between DDR Memory and MSS Internal Memory	243
Figure 126	HPDMA Transfers Data Between SDR Memory and MSS Internal Memory	243
Figure 127	HPDMA Driver User Guide	244
Figure 128	HPDMA Examples	245
Figure 129	PDMA Interfacing with AHB Bus Matrix	264
Figure 130	PDMA Internal Architecture	265
Figure 131	Flow of Ping-Pong Operation on DMA Channel	267
Figure 132	Enable PDMA	271
Figure 133	PDMA AHB Bus Master Matrix Configuration	272
Figure 134	PDMA Transfers Data Between FIC and MSS Memory	272
Figure 135	PDMA Signals	273
Figure 136	PDMA Driver User Guide	273
Figure 137	PDMA Examples	274
Figure 138	MSS Showing a USB OTG Controller	285
Figure 139	USB OTG Controller in SmartFusion2	286
Figure 140	Block Diagram for Connections between USB Controller and ULPI PHY through MSS	288
Figure 141	Block Diagram for Connections Between USB Controller and UTMI PHY through FPGA Fabric	290
Figure 142	Basic USB Flow Diagram when USB Controller is in Host Mode	292
Figure 143	Basic USB Flow Diagram when USB Controller is in USB Device/Peripheral Mode	293
Figure 144	Basic USB Flow Diagram when USB Controller is in OTG Mode	295
Figure 145	LPM State Transition Diagram	296
Figure 146	MSS Configurator with USB and GPIO Macros Enabled	300
Figure 147	MSS USB Configurator with ULPI Interface Settings	301
Figure 148	MSS USB Configurator with UTMI Interface Settings	302
Figure 149	MSS GPIO Configurator with GPIO Settings for External USB PHY Reset	303
Figure 150	I/O Editor Configurator with Settings for External USB PHY Reset Pin Mapping	303
Figure 151	Firmware Catalog with MSS USB Firmware Drivers and Sample Class Drivers	304
Figure 152	MSS Showing a TSEMAC	374
Figure 153	TSEMAC Block Diagram	375
Figure 154	RMII, RGMII, RTBI, RevMII, SMII Derived from Available Protocols by Appropriate Wrapper in Fabric	380
Figure 155	TBI Brought to Fabric for EPCS Soft IP for SGMII Interface	380
Figure 156	External PHY Interface Selection in MSS EMAC Configurator	384
Figure 157	Line Speed Selection in MSS EMAC Configurator	385
Figure 158	External PHY Management Interface Selections in MSS EMAC Configurator	386
Figure 159	MSS Ethernet Configurator with TBI interface	387
Figure 160	SGMII Interface Signals: TBI to SERDES	387
Figure 161	I/O Editor With SGMII and PHY Ports	388
Figure 162	SECEDED Configurator with Ethernet TX RAM and Ethernet RX RAM Configuration Options	389
Figure 163	Firmware Catalog Showing the Generation of Sample Project for TSEMAC	390
Figure 164	CoreMACFilter Interaction with MSS MAC and GMII Ethernet PHY	435
Figure 165	CoreMACFilter interaction with MSS MAC and SGMII Ethernet PHY	435
Figure 166	CAN Controller Block Diagram	436
Figure 167	Transmit Message Buffers	438
Figure 168	Receive Message Buffers	439
Figure 169	Interrupt Generation	441
Figure 170	CAN Configurator GUI	442
Figure 171	Main Connection Options - Either MSIO or Fabric	443

Figure 172	Extra Connection to Fabric and GPIO Options	444
Figure 173	Enabling EDAC for the CAN from the SECCDED Configurator	445
Figure 174	Enabling CAN Controller With MSS Configurator	446
Figure 175	CAN Signals	446
Figure 176	Firmware Driver Enable and Generate	447
Figure 177	Automatic Bit Rates Detection Flow Chart	448
Figure 178	MSS Showing MMUART Peripherals	470
Figure 179	MMUART Block Diagram	471
Figure 180	Synchronous and Asynchronous Mode Topologies	475
Figure 181	Sample Time Correction with Fractional Baud Rate	476
Figure 182	Example with Fractional Baud Rate of 4.5	477
Figure 183	Example with Fractional Baud Rate of 2 and 3/64 th	477
Figure 184	Synchronous Input and Adaptation to Internal Baud Clocking	478
Figure 185	Bi-Directional Synchronous Clock Configuration Options	479
Figure 186	Input Filtering Circuit and Timing for GLR=4 (Pulses Less than 4 APB Clock Cycles Filtered Out)	480
Figure 187	LIN Header	480
Figure 188	LIN Break Field Width => 11 Tbit Count Interrupt	481
Figure 189	LIN PID Parity Error Interrupt	481
Figure 190	LIN Receive FSM	482
Figure 191	RZI Modulation	482
Figure 192	RX RZI-to-NRZ Demodulation	483
Figure 193	Tx NRZ-to-RZI Modulation	483
Figure 194	9-Bit Format	484
Figure 195	Single Wire Error Signal Timing when EERR=1	485
Figure 196	Transmit Mode TE Output Enable Timing when EERR=1	485
Figure 197	Receive Mode TE Output Enable and NACK Timing when EERR=1	486
Figure 198	Enable MMUART	486
Figure 199	MSS MMUART Configurator	487
Figure 200	MMUART Interface Signals	487
Figure 201	MMUART Driver User Guide	488
Figure 202	MMUART Sample Project	489
Figure 203	Setup to Communicate With Host PC Through MMUART Interface - Block Diagram	489
Figure 204	Microcontroller Subsystem Showing SPI Peripherals	504
Figure 205	SPI Controller Block Diagram	505
Figure 206	Motorola SPI Mode 0	508
Figure 207	Motorola SPI Mode 0 Multiple Frame Transfer	508
Figure 208	Motorola SPI Mode 1	509
Figure 209	Motorola SPI Mode 2	509
Figure 210	Motorola SPI Mode 3	509
Figure 211	Write Operation Timing	511
Figure 212	Read Operation Timing	511
Figure 213	Page Program Timing	512
Figure 214	National Semiconductor MICROWAVE Single Frame Transfer	513
Figure 215	National Semiconductor MICROWIRE Multiple Frame Transfer	513
Figure 216	TI Synchronous Serial Single Frame Transfer	514
Figure 217	TI Synchronous Serial Multiple Frame Transfer	514
Figure 218	SPE Command/Data Format	515
Figure 219	Enable SPI	520
Figure 220	MSS SPI Configurator - Connection Type IO	521
Figure 221	MSS SPI Configurator - Connection Type Fabric	522
Figure 222	SPI Interface Signals - Connection Type IO	522
Figure 223	SPI Interface Signals - Connection Type Fabric	523
Figure 224	SPI Driver User's Guide	523
Figure 225	SPI Sample Project	524
Figure 226	Interfacing External SPI Flash to MSS SPI_0-Block Diagram	525
Figure 227	Interfacing External SPI Slave Device Using MSS SPI Routing Through Fabric Block Diagram	526
Figure 228	Microcontroller Subsystem Showing I ² C Peripherals	538
Figure 229	I ² C Block Diagram	539

Figure 230	8-bit Data Transfer Cycle	540
Figure 231	PMBus and SMBus Devices Interface	543
Figure 232	Enable I ² C	544
Figure 233	MSS I2C Configurator	545
Figure 234	I ² C Interface Signals	545
Figure 235	I ² C Driver User's Guide	546
Figure 236	I ² C Sample Project	547
Figure 237	Interfacing External EEPROM to MSS I2C_0 - Block Diagram	548
Figure 238	I ² C Loopback Block Diagram	550
Figure 239	GPIO Connected on APB Slave in MSS	562
Figure 240	GPIO, IOMUX, and MSIO	563
Figure 241	MSS GPIO Block Diagram	564
Figure 242	GPIO Configuration Register (GPIO_X_CFG)	565
Figure 243	Enable GPIO in MSS Configurator	568
Figure 244	MSS GPIO Configurator	569
Figure 245	Configuring GPIO Byte-Wise Reset	570
Figure 246	Configuring GPIOs as Input, Output, Tristate, or Bi-directional	571
Figure 247	Connectivity Preview	572
Figure 248	GPIO Signals	572
Figure 249	GPIO Loopback Diagram	574
Figure 250	Interfacing of COMM_BLK with AHB Bus Matrix	593
Figure 251	Interfacing of COMM_BLK with System Controller	594
Figure 252	System Services Driver in Firmware Core Configurator	596
Figure 253	System Services Driver Folder Hierarchy	596
Figure 254	Microcontroller Subsystem Showing RTC	601
Figure 255	RTC Block Diagram	602
Figure 256	Enabling RTC in the Libero SOC Design MSS Configurator	605
Figure 257	RTC Configuration Window	606
Figure 258	RTC Signals	606
Figure 259	RTC Driver User Guide	607
Figure 260	RTC Examples	608
Figure 261	MSS Showing Timer Peripherals	614
Figure 262	Timer Block Diagram	615
Figure 263	Block Diagram 32-Bit Mode	618
Figure 264	Block Diagram 64-Bit Mode	618
Figure 265	Timer Driver User Guide	619
Figure 266	Generating Sample Project	621
Figure 267	Microcontroller Subsystem Showing Watchdog Timer	629
Figure 268	Block Diagram for the Watchdog Timer	630
Figure 269	Watchdog Timer Counter	632
Figure 270	Enabling Watchdog Timer in the Libero SOC Design MSS Configurator	634
Figure 271	Watchdog Timer Configuration Window	635
Figure 272	Watchdog Timer Signals	635
Figure 273	RTC Driver User Guide	636
Figure 274	Watchdog Timer Examples	637
Figure 275	Reset Signals Distribution in SmartFusion2 Devices	642
Figure 276	Conceptual Block Diagram of Power-On Reset Generation	643
Figure 277	Power on Reset Delay Configuration	644
Figure 278	SYSRESET Macro	644
Figure 279	Power up to Functional Time Sequence Diagram	645
Figure 280	VDD Power-up to Functional Time Design Setup	647
Figure 281	VDD Power-up to functional timing diagram	648
Figure 282	VDD Power-Up to Functional Time Flow	650
Figure 283	DEVRST_N Power-Up to Functional Timing	651
Figure 284	DEVRST_N Power-up to Functional Time Flow	653
Figure 285	Functional Block Diagram of Reset Controller During Power-On Reset	654
Figure 286	SYSRESET_N Generation	655
Figure 287	Functional Block Diagram of Reset Controller During SYSRESET_N	655
Figure 288	Reset Controller With Only Block Level Resets	656

Figure 289	M3_SYS_RESET_N Generation	657
Figure 290	MDDR_AXI_RESET_N Generation	657
Figure 291	MDDR_APB_RESET_N Generation	658
Figure 292	WDOG_RESET_N Generation	658
Figure 293	FIC_2_APB_M_PRESET_N Generation	659
Figure 294	MSS GPIO_OUT Reset Generation	660
Figure 295	Block Level Reset Generation	660
Figure 296	MSS_READY Signal Generation	661
Figure 297	CoreResetP Connectivity with Peripheral Resets	662
Figure 298	CoreResetP Connectivity with SERDES_IF Block	663
Figure 299	Timing for Reset Signals Initiated by the Assertion of POWER_N_RESET_N	664
Figure 300	Timing for Reset Signals Initiated by the Assertion of FIC_2_APB_M_PRESET_N	664
Figure 301	Timing for Reset Signals Initiated by the Assertion of EXT_RESET_IN_N	665
Figure 302	Timing for Reset Signals Initiated by the Assertion of USER_FAB_RESET_IN_N	665
Figure 303	Ramp Delay Configuration	666
Figure 304	Configuring Reset	666
Figure 305	Configuring Reset	667
Figure 306	Connecting Fabric Logic	667
Figure 307	Initialization Sub-system with CoreResetP Soft IP	668
Figure 308	Initialization Sub-system for FIC Sub-systems	669
Figure 309	Register Write Protect	670
Figure 310	Field Write Protect	671
Figure 311	Bit Write Protect	671
Figure 312	RW-P Type	672
Figure 313	RW Type	673
Figure 314	RO Type	673
Figure 315	RO-P Type	674
Figure 316	RO-U Type	674
Figure 317	Register Lock Bit Settings	674
Figure 318	Lock Bit Configuration File	675
Figure 319	The FIIC Connection to AHB Bus Matrix	738
Figure 320	Block Diagram for Fabric Interface Interrupt Controller	739
Figure 321	Combinational Circuit for Mapping MSS Interrupts to a MSS_INT_M2F	739
Figure 322	Configure FIIC in the MSS Configurator	741
Figure 323	FIIC Configurator	742
Figure 324	Fabric to the MSS Interrupt	743
Figure 325	MSS to Fabric Interrupt	745
Figure 326	The FIC Connection to the AHB Bus Matrix	757
Figure 327	Fabric Interface Controller Block Diagram	758
Figure 328	Fabric Interface Controller Top-Level View	761
Figure 329	AHB-Lite Bus Signals from FIC to the Fabric Slave for a Write Transaction in Bypass Mode	763
Figure 330	AHB-Lite Bus Signals from FIC to the Fabric Slave for a Read Transaction in Bypass Mode	763
Figure 331	AHB-Lite Bus Signals from FIC to the Fabric Slave for a Write Transaction in Synchronous Pipelined Mode 764	764
Figure 332	AHB-Lite Bus Signals from FIC to the Fabric Slave for a Read Transaction in Synchronous Pipelined Mode 764	764
Figure 333	AHB-Lite Bus Signals from Fabric Master to FIC for a Write Transaction in Bypass Mode	765
Figure 334	AHB-Lite Bus Signals from Fabric Master to FIC for a Read Transaction in Bypass Mode	765
Figure 335	AHB-Lite Bus Signals from Fabric Master to FIC for a Write Transaction in Synchronous Pipelined Mode 766	766
Figure 336	AHB-Lite Bus Signals from Fabric Master to FIC for a Read Transaction in Synchronous Pipelined Mode 766	766
Figure 337	MSS Configurator	767
Figure 338	FIC Configurator	768
Figure 339	MSS to FPGA Fabric Interface Core	768
Figure 340	Advanced Options Configuration	769
Figure 341	FPGA Fabric Address Regions (MSS Master View)	769
Figure 342	Master/AHB-Lite Memory Space Configuration – 16 MB per Slot	769
Figure 343	Master/AHB-Lite Memory Space Configuration – 256 MB per Slot	770

Figure 344	Master/AHB-Lite Master Access Configuration	770
Figure 345	FIC Master/AHB-Lite Subsystem	772
Figure 346	Master/APB Address Configuration	773
Figure 347	Master/APB Slave Slots Configuration	773
Figure 348	FIC Master/APB Subsystem	774
Figure 349	Clocking Scheme for Synchronous Communication Between the MSS and the FPGA Fabric ..	775
Figure 350	MSS CCC FIC Clock Configuration	776
Figure 351	Fabric Clocks Configuration	777
Figure 352	Configure the MSS Reset Sub-Block	778
Figure 353	AHB-Lite Slaves in the FPGA Fabric Connected to the MSS Master	779
Figure 354	APB Slaves in the FPGA Fabric Connected to the MSS	780
Figure 355	FPGA System with the MSS Slave and the Fabric Master	781
Figure 356	Fabric APB Master with MSS as Slave	782
Figure 357	APB Configuration Interface and Subsystems Connectivity with MSS Master	784
Figure 358	Configure FIC_2 in MSS Configurator	787
Figure 359	FIC_2 Configurator	788
Figure 360	FIC_2 Configuration for MSS DDR	788
Figure 361	FIC_2 Configuration for MSS DDR, FDDR, and SERDES	789
Figure 362	MSS DDR Design with APB Configuration Interface	790
Figure 363	Top-Level Components with APB Configuration Interface Signals	791
Figure 364	Interfacing of CoreSF2Config Mirrored APB Slave with SERDES_IF Block	791
Figure 365	EDAC in Write Mode	792
Figure 366	EDAC in Read Mode (Reading From Memory)	792
Figure 367	EDAC in Read Mode (Reading From Memory)	794
Figure 368	EDAC in Read Mode (Reading From Memory)	795

Tables

Table 1	Cortex-M3 Processor Exceptions	8
Table 2	Cortex-M3 Processor Interrupts	8
Table 3	Signal Multiplexing	14
Table 4	Port Details of the Cortex-M3-Subsystem	15
Table 5	MPU Configuration Register	17
Table 6	SysTick Configuration Register	17
Table 7	Summary of Processor Mode, Execution Privilege Level, and Stack Use Options	21
Table 8	Core Register Set Summary	21
Table 9	PSR Combinations and Attributes	23
Table 10	Application Program Status Register	23
Table 11	IPSR Bit Assignments	23
Table 12	EPSR Bit Assignments	24
Table 13	PRIMASK Register Bit Assignments	25
Table 14	FAULT Register Bit Assignments	26
Table 15	BASEPRI Register Bit Assignments	26
Table 16	Control Register Bit Assignments	27
Table 17	Memory Access Behavior	30
Table 18	Memory Region Shareability and Cache Policies	31
Table 19	SRAM Memory Bit-banding Regions	32
Table 20	Peripheral Memory Bit-banding Regions	32
Table 21	CMSIS Functions for Exclusive Access Instructions	37
Table 22	Properties of the Different Exception Types	39
Table 23	Exception Return Behavior	43
Table 24	Faults	43
Table 25	Fault Status and Fault Address Registers	45
Table 26	Cortex-M3 Processor Instructions	47
Table 27	CMSIS Functions to Generate some Cortex-M3 Processor instructions	50
Table 28	CMSIS Functions to Access the Special Registers	51
Table 29	Condition Code Suffixes	57
Table 30	Memory Access Instructions	58
Table 31	Offset Ranges	60
Table 32	Offset Ranges	63
Table 33	Data Processing Instructions	68
Table 34	Multiply and Divide Instructions	77
Table 35	Packing and Unpacking Instructions	81
Table 36	Branch and Control Instructions	83
Table 37	Branch Ranges	84
Table 38	Miscellaneous Instructions	89
Table 39	Core Peripheral Register Regions	95
Table 40	NVIC Register Summary	95
Table 41	CMSIS Access NVIC Functions	96
Table 42	NVIC_IUSER Bit Assignments	97
Table 43	NVIC_ICER Bit Assignments	97
Table 44	NVIC_ISPR Bit Assignments	98
Table 45	NVIC_ICPR bit assignments	98
Table 46	NVIC_IABR Bit Assignments	99
Table 47	NVIC_IPR Bit Assignments	99
Table 48	STIR Bit Assignments	100
Table 49	CMSIS Functions for NVIC Control	101
Table 50	Summary of the System Control Block Registers	102
Table 51	ACTLR Bit Assignments	103
Table 52	CPUID register Bit Assignments	103
Table 53	ICSR Bit Assignments	104
Table 54	VTOR Bit Assignments	106

Table 55	AIRCR Bit Assignments	107
Table 56	Priority Grouping	107
Table 57	SCR Bit Assignments	108
Table 58	CCR Bit Assignments	109
Table 59	System Fault Handler Priority Fields	110
Table 60	SHPR1 Bit Assignments	111
Table 61	SHPR2 Bit Assignments	111
Table 62	SHPR3 Bit Assignments	111
Table 63	SHCSR Bit Assignments	112
Table 64	MMFSR Bit Assignments	114
Table 65	BFSR Bit Assignments	115
Table 66	UFSR Bit Assignments	116
Table 67	HFSR Bit Assignments	118
Table 68	MMFAR Bit Assignments	118
Table 69	BFAR Bit Assignments	118
Table 70	AFSR Bit Assignments	119
Table 71	System Timer Registers Summary	119
Table 72	SYST_CTRL Register Bit Assignments	120
Table 73	SYST_RVR Register Bit Assignments	120
Table 74	SYST_CVR Register Bit Assignments	121
Table 75	SYST_CALIB Register Bit Assignments	121
Table 76	Memory Attributes Summary	123
Table 77	MPU Registers Summary	123
Table 78	MPU_TYPE Register Bit Assignments	124
Table 79	MPU_CTRL Register Bit Assignments	124
Table 80	MPU_RNR Bit Assignments	126
Table 81	MPU_RBAR Bit Assignments	126
Table 82	MPU_RASR Bit Assignments	127
Table 83	Example SIZE Field Values	128
Table 84	TEX, C, B, and S Encoding	128
Table 85	Cache Policy for Memory Attribute Encoding	129
Table 86	AP Encoding	129
Table 87	Memory Region Attributes for a Microcontroller	132
Table 88	Default (eNVM Remapped Mode)	135
Table 89	eSRAM Remapped Mode (Memory Map)	135
Table 90	DDR Remap	136
Table 91	Data Path for Various Maps	137
Table 92	System Registers for Cache Operations	144
Table 93	eNVM Address Locations	146
Table 94	Memory Organization	147
Table 95	Data Retention Time	148
Table 96	AHBL Address Map to NVM	150
Table 97	Command (CMD) Register	151
Table 98	Command Table	151
Table 99	User Protection Regions	162
Table 100	Special Purpose Storage Regions	163
Table 101	Special Purpose Storage Regions for M2S060, M2S090, and M2S150 Devices	164
Table 102	Available APIs for eNVM	174
Table 103	SYSREG Control Registers	174
Table 104	ENVM_CR	175
Table 105	SW_ENVMREMAPSIZE	177
Table 106	ENVM_REMAP_BASE_CR	177
Table 107	ENVM_REMAP_FAB_CR	178
Table 108	ENVM_PROTECT_USER	178
Table 109	ENVM_STATUS	179
Table 110	ENVM_SR	180
Table 111	eNVM Control Registers Base Address	180
Table 112	Control Registers Description	180
Table 113	Status Register Bit Definitions	184

Table 114	NV_FREQRNG Calculations at Different M3_CLK Frequencies for All SmartFusion2 Devices . .	185
Table 115	NV_PAGE_STATUS	185
Table 116	INTEN[10:0]	185
Table 117	CLRHint[2:0]	186
Table 118	eSRAM Block Sizes and Address Ranges	188
Table 119	SRAM Organization in SECDED-ON Mode	190
Table 120	SRAM Organization in SECDED-OFF Mode	190
Table 121	Wait States in Different Operation Modes	192
Table 122	SYSREG Control Registers	198
Table 123	ESRAM_CR	199
Table 124	ESRAM_MAX_LAT	200
Table 125	eSRAM Maximum Latency Values	200
Table 126	ESRAM_PIPELINE_CR	201
Table 127	ESRAM0_EDAC_CNT	201
Table 128	ESRAM1_EDAC_CNT	201
Table 129	ESRAM0_EDAC_ADR	201
Table 130	ESRAM1_EDAC_ADR	202
Table 131	MM0_1_2_SECURITY	202
Table 132	MM4_5_DDR_FIC_SECURITY/MM4_5_FIC64_SECURITY	203
Table 133	MM3_6_7_8_SECURITY	204
Table 134	MM9_SECURITY	204
Table 135	EDAC_SR	205
Table 136	CLR_EDAC_COUNTERS	206
Table 137	EDAC_IRQ_ENABLE_CR	207
Table 138	EDAC_CR	208
Table 139	AHB Bus Matrix Connectivity	211
Table 140	Fixed Priority Masters	218
Table 141	WRR Masters	218
Table 142	Pure Round Robin and Fixed Priority Arbitration Scenario for eSRAM1	220
Table 143	WRR and Fixed Priority Arbitration Scenario for eNVM_0	222
Table 144	WRR Arbitration Scenario for eSRAM_0 slave	222
Table 145	Decoding of Master Access to the Fabric Slaves	224
Table 146	Pairing of Masters and Slaves	232
Table 147	AHB Bus Matrix Register Map	235
Table 148	MSS HPDMA APIs	244
Table 149	HPDMA Register Map	246
Table 150	HPDMAEDR_REG	247
Table 151	HPDMAD0SAR_REG	251
Table 152	HPDMAD1SAR_REG	251
Table 153	HPDMAD2SAR_REG	251
Table 154	HPDMAD3SAR_REG	251
Table 155	HPDMAD0DAR_REG	252
Table 156	HPDMAD1DAR_REG	252
Table 157	HPDMAD2DAR_REG	252
Table 158	HPDMAD3DAR_REG	252
Table 159	HPDMAD0CR_REG	253
Table 160	HPDMAD1CR_REG	254
Table 161	HPDMAD2CR_REG	255
Table 162	HPDMAD3CR_REG	256
Table 163	HPDMAD0SR_REG	257
Table 164	HPDMAD1SR_REG	257
Table 165	HPDMAD2SR_REG	258
Table 166	HPDMAD3SR_REG	259
Table 167	HPDMAD0PTR_REG	259
Table 168	HPDMAD1PTR_REG	260
Table 169	HPDMAD2PTR_REG	260
Table 170	HPDMAD3PTR_REG	261
Table 171	HPDMAICR_REG	261
Table 172	HPDMADR_REG	262

Table 173	SYSREG Control Registers	263
Table 174	RATIOHILO Field Definition	268
Table 175	Port List	268
Table 176	MSS PDMA APIs	273
Table 177	SmartFusion2 SoC FPGA PDMA Register Map	275
Table 178	Ratio_HIGH_LOW	278
Table 179	BUFFER_STATUS	278
Table 180	CHANNEL_x_CONTROL	279
Table 181	PERIPHERAL_SEL	280
Table 182	CHANNEL_x_STATUS	281
Table 183	CHANNEL_x_BUFFER_A_SRC_ADDR	281
Table 184	CHANNEL_x_BUFFER_A_DST_ADDR	281
Table 185	CHANNEL_x_BUFFER_A_TRANSFER_COUNT	282
Table 186	CHANNEL_x_BUFFER_B_SRC_ADDR	282
Table 187	CHANNEL_x_BUFFER_B_DST_ADDR	282
Table 188	CHANNEL_x_BUFFER_B_TRANSFER_COUNT	282
Table 189	SYSREG Control Registers	283
Table 190	ULPI Interface Signals at SmartFusion2 External I/Os	287
Table 191	USB IO Group Availability	288
Table 192	UTMI+Interface Signals at Fabric Interface in SmartFusion2 Device	288
Table 193	Response to LPM Transaction as Peripheral	297
Table 194	APIs available in USB Firmware Drivers and Description of CIL	304
Table 195	Device Mode Class Driver APIs (LDL)	306
Table 196	Functional Descriptions of Callback APIs	306
Table 197	SOFTRESET_REG Bit for USB Controller Soft Reset	308
Table 198	Common Register Set Description	309
Table 199	FADDR_REG(0x40043000)	310
Table 200	POWER_REG (0x40043001)	310
Table 201	TX_IRQ_REG (0x40043002)	311
Table 202	RX_IRQ_REG (0x40043004)	311
Table 203	TX_IRQ_EN_REG (0x40043006)	312
Table 204	RX_IRQ_EN_REG (0x40043008)	312
Table 205	USB_IRQ_REG (0x4004300A)	312
Table 206	USB_IRQ_EN_REG (0x4004300B)	313
Table 207	FRAME_REG (0x4004300C)	313
Table 208	INDEX_REG (0x4004300E)	313
Table 209	TEST_MODE_REG (0x4004300F)	314
Table 210	Operating Speed	314
Table 211	Indexed Register Set Description	315
Table 212	TX_MAX_P_REG	317
Table 213	CSR0L_REG (Peripheral)	317
Table 214	CSR0L_REG (Host)	318
Table 215	CSR0H_REG (Peripheral)	319
Table 216	CSR0H_REG (Host)	319
Table 217	TX_CSRL_REG (Peripheral)	320
Table 218	TX_CSRL_REG (Host)	320
Table 219	TX_CSRH_REG (Peripheral)	321
Table 220	TX_CSRH_REG (Host)	322
Table 221	RX_MAX_P_REG	323
Table 222	RX_CSRL_REG (Peripheral)	324
Table 223	RX_CSRL_REG (Host)	324
Table 224	RX_CSRH_REG (Peripheral)	325
Table 225	RxPktReady Bit Cleared	326
Table 226	RX_CSRH_REG (Host)	326
Table 227	COUNT0_REG	327
Table 228	RX_COUNT_REG	327
Table 229	TYPE0_REG	327
Table 230	TX_TYPE_REG	328
Table 231	NAK_LIMIT0_REG	328

Table 232	TX_INTERVAL_REG	328
Table 233	Polling Intervals for Transfer Types	329
Table 234	RX_TYPE_REG	329
Table 235	RX_INTERVAL_REG	329
Table 236	CONFIG_DATA_REG	330
Table 237	FIFO_SIZE_REG	330
Table 238	FIFO Registers	330
Table 239	EPx_FIFO_REG (0x400430YZ)	331
Table 240	Additional Control and Status Registers (OTG, Dynamic FIFO, and Version)	331
Table 241	DEV_CTRL_REG (0x40043060)	334
Table 242	Encoding of VBus Level	334
Table 243	MISC_REG (0x40043061)	335
Table 244	TX_FIFO_SIZE_REG (0x40043062)	335
Table 245	RX_FIFO_SIZE_REG (0x40043063)	336
Table 246	TX_FIFO_ADD_REG (0x40043064)	336
Table 247	Start Address of Transmit Endpoint	336
Table 248	RX_FIFO_ADD_REG (0x40043066)	337
Table 249	VBUS_CSR_REG (write only) (0x40043068)	337
Table 250	VBUS_CSR_REG (read only) (0x40043068)	337
Table 251	HW_VERSION_REG (0x4004306C)	337
Table 252	ULPI and Configuration Registers	338
Table 253	ULPI_VBUS_CTRL_REG (0x40043070)	339
Table 254	ULPI_CARKIT_CTRL_REG (0x40043071)	339
Table 255	ULPI_IRQ_MASK_REG (0x40043072)	340
Table 256	ULPI_IRQ_SRC_REG (0x40043073)	340
Table 257	ULPI_DATA_REG (0x40043074)	340
Table 258	ULPI_ADDR_REG (0x40043075)	340
Table 259	ULPI_REG_CTRL (0x40043076)	341
Table 260	ULPI_RAW_DATA_REG (0x40043077) (Asynchronous)	341
Table 261	ULPI_RAW_DATA_REG (0x40043077) (Synchronous)	341
Table 262	Encoded UTMI Event Signals	342
Table 263	EP_INFO_REG (0x40043078)	342
Table 264	RAM_INFO_REG (0x40043079)	342
Table 265	LINK_INFO_REG (0x4004307A)	342
Table 266	VP_LEN_REG (0x4004307B)	343
Table 267	HS_EOF1_REG (0x4004307C)	343
Table 268	FS_EOF1_REG (0x4004307D)	343
Table 269	LS_EOF1_REG (0x4004307E)	343
Table 270	SOFT_RESET_REG (0x4004307F)	343
Table 271	Endpoint0 Control and Status Registers	344
Table 272	Endpoint1 Control and Status Registers	345
Table 273	Endpoint2 Control and Status Registers	345
Table 274	Endpoint3 Control and Status Registers	346
Table 275	Endpoint4 Control and Status Registers	347
Table 276	EPx_TX_MAX_P_REG	348
Table 277	EPx_RX_COUNT_REG	348
Table 278	EPx_RX_MAX_P_REG	348
Table 279	EPx_TX_TYPE_REG	349
Table 280	EPx_TX_INTERVAL_REG	349
Table 281	EPx_RX_TYPE_REG	349
Table 282	EPx_RX_INTERVAL_REG	350
Table 283	EPx_FIFO_SIZE_REG	350
Table 284	Extended Registers Description	351
Table 285	EPx_RQ_PKT_COUNT_REG (0x40043XYZ)	352
Table 286	RX_DPKT_BUF_DIS_REG (0x40043340)	352
Table 287	TX_DPKT_BUF_DIS_REG (0x40043342)	353
Table 288	C_T_UCH_REG (0x40043344)	353
Table 289	C_T_HHSRTN_REG (0x40043346)	353
Table 290	C_T_HSBT_REG (0x40043348)	353

Table 291	Turnaround Timeout Period Settings	354
Table 292	DMA_REGISTER Description	354
Table 293	DMA_INT_REG (0x40043200)	356
Table 294	CHx_DMA_CTRL_REG (0x40043204)	356
Table 295	CHx_DMA_ADDR_REG (0x40043208)	357
Table 296	CHx_DMA_COUNT_REG (0x4004320C)	358
Table 297	Additional Multipoint CSR Description	358
Table 298	EPx_TX_FUNC_ADDR_REG (0x40043080)	364
Table 299	EPx_RX_FUNC_ADDR_REG	364
Table 300	EPx_TX_HUB_ADDR_REG (0x40043082)	364
Table 301	EPx_RX_HUB_ADDR_REG	365
Table 302	EPx_TX_HUB_PORT_REG (0x40043083)	365
Table 303	EPx_RX_HUB_PORT_REG	365
Table 304	Link Power Management Register Descriptions	366
Table 305	LPM_ATTR_REG (0x40043360)	367
Table 306	LPM_CTRL_REG (0x40043362)(Peripheral)	367
Table 307	LPM_CTRL_REG (0x40043362) (Host)	368
Table 308	LPM_INTR_EN_REG (0x40043363)	368
Table 309	LPM_INTR_REG (0x40043364)(Peripheral Mode)	369
Table 310	LPM_INTR_REG (0x40043364) (Host Mode)	369
Table 311	LPM_FADDR_REG (0x40043365)	370
Table 312	MASTER_WEIGHT1_CR	371
Table 313	USB_IO_INPUT_SEL_CR	372
Table 314	USB_CR	372
Table 315	USB_EDAC_CNT	372
Table 316	USB_EDAC_ADR	373
Table 317	USB_SR	373
Table 318	EDAC_SR	373
Table 319	CLR_EDAC_COUNTERS	373
Table 320	MII Ports	378
Table 321	GMII Ports	378
Table 322	TBI Ports	379
Table 323	Tx/Rx Descriptor	381
Table 324	PacketStartAddr	381
Table 325	Packet Size	381
Table 326	Next Descriptor	382
Table 327	TSEMAC Firmware Drivers for Initialization and Configuration	391
Table 328	TSEMAC Firmware Drivers for Ethernet PHY Management	391
Table 329	TSEMAC Firmware Drivers for Transmit and Receive Operations	391
Table 330	TSEMAC Firmware Drivers for Reading Status and Statistics	392
Table 331	MAC_CR Register in SYSREG Block	392
Table 332	EMAC M-AHB Register Map	393
Table 333	EMAC PE-MCXMAC Register Map	393
Table 334	EMAC A-MCXFIFO Register Map	394
Table 335	EMAC PE-MSTAT Transmit and Receive Counters Register Map	395
Table 336	EMAC PE-MSTAT Receive Counters Register Map	395
Table 337	EMAC PE-MSTAT Transmit Counters Register Map	396
Table 338	EMAC M-SGMII Register Map	397
Table 339	DMA_TX_CTRL	399
Table 340	DMA_TX_DESC	399
Table 341	DMA_TX_STATUS	399
Table 342	DMA_RX_CTRL	400
Table 343	DMA_RX_DESC	400
Table 344	DMA_RX_STATUS	400
Table 345	DMA_IRQ_MASK	400
Table 346	DMA_IRQ	401
Table 347	CFG1	401
Table 348	CFG2	402
Table 349	IFG	403

Table 350	HALF_DUPLEX	404
Table 351	MAX_FRAME_LENGTH	405
Table 352	TEST	405
Table 353	MII_CONFIG	405
Table 354	MII_COMMAND	406
Table 355	MII_ADDRESS	407
Table 356	MII_CTRL	407
Table 357	MII_STATUS	407
Table 358	MII_INDICATORS	407
Table 359	INTERFACE_CTRL	407
Table 360	INTERFACE_STATUS	409
Table 361	STATION_ADDRESS1	409
Table 362	STATION_ADDRESS2	410
Table 363	FIFO_CFG0	410
Table 364	FIFO_CFG1	411
Table 365	FIFO_CFG2	411
Table 366	FIFO_CFG3	412
Table 367	FIFO_CFG4	413
Table 368	FIFO_CFG5	413
Table 369	FIFO_RAM_ACCESS0	414
Table 370	FIFO_RAM_ACCESS1	415
Table 371	FIFO_RAM_ACCESS2	415
Table 372	FIFO_RAM_ACCESS3	415
Table 373	FIFO_RAM_ACCESS4	415
Table 374	FIFO_RAM_ACCESS5	415
Table 375	FIFO_RAM_ACCESS6	416
Table 376	FIFO_RAM_ACCESS7	416
Table 377	TR64	416
Table 378	TR127	416
Table 379	TR255	416
Table 380	TR511	417
Table 381	TR1K	417
Table 382	TRMAX	417
Table 383	TRMGV	417
Table 384	RBYT	417
Table 385	RPKT	418
Table 386	RFCS	418
Table 387	RMCA	418
Table 388	RBCA	418
Table 389	RXCF	418
Table 390	RXPf	418
Table 391	RXUO	419
Table 392	RALN	419
Table 393	RFLR	419
Table 394	RCDE	419
Table 395	RCSE	419
Table 396	RUND	420
Table 397	ROVR	420
Table 398	RFRG	420
Table 399	RJBR	420
Table 400	RDRP	420
Table 401	TBYT	421
Table 402	TPKT	421
Table 403	TMCA	421
Table 404	TBCA	421
Table 405	TXPF	421
Table 406	TDFR	422
Table 407	TEDF	422
Table 408	TSCL	422

Table 409	TMCL	422
Table 410	TLCL	422
Table 411	TXCL	422
Table 412	TNCL	423
Table 413	TPFH	423
Table 414	TDRP	423
Table 415	TJBR	423
Table 416	TFCS	423
Table 417	TXCF	423
Table 418	TOVR	424
Table 419	TUND	424
Table 420	TFRG	424
Table 421	CAR1	424
Table 422	CAR2	425
Table 423	CAM1	426
Table 424	CAM2	427
Table 425	SGMII CONTROL	429
Table 426	SGMII STATUS	429
Table 427	AN SGMII ADVERTISEMENT	430
Table 428	AN LINK PARTNER BASE PAGE ABILITY	430
Table 429	AN EXPANSION	431
Table 430	AN NEXT PAGE TRANSMIT	431
Table 431	AN NEXT PAGE TRANSMIT	431
Table 432	EXTENDED STATUS	432
Table 433	JITTER DIAGNOSTICS	433
Table 434	TBI CONTROL	433
Table 435	CAN BUS Interface	437
Table 436	Test Modes	442
Table 437	Summary of Different Valid CAN Connections	443
Table 438	APB to SRAM Address Mapping	449
Table 439	CAN SYSREG Control Registers	451
Table 440	CAN Controller Soft Reset Bit in the SOFT_RESET_CR Register	452
Table 441	Summary of CAN Controller Registers	452
Table 442	CAN_CONFIG	454
Table 443	CAN_COMMAND	455
Table 444	TX_MSG0_CTRL_CMD	456
Table 445	TX_MSG0_ID	457
Table 446	TX_MSG0_DATA_HIGH	457
Table 447	TX_MSG0_DATA_LOW	457
Table 448	Transmit Message1 to Transmit Message31 Registers Description	458
Table 449	TX_BUF_STATUS	459
Table 450	RX_MSG0_CTRL_CMD	460
Table 451	RX_MSG0_ID	461
Table 452	RX_MSG0_DATA_HIGH	461
Table 453	RX_MSG0_DATA_LOW	462
Table 454	RX_MSG0_AMR	462
Table 455	RX_MSG0_ACR	462
Table 456	RX_MSG0_AMR_DATA	462
Table 457	RX_MSG0_ACR_DATA	462
Table 458	Receive Message1 to Receive Message 31 and ECR Registers Description	463
Table 459	RX_BUF_STATUS	464
Table 460	ECR	465
Table 461	ERROR_STATUS	466
Table 462	INT_ENABLE	466
Table 463	INT_STATUS	467
Table 464	MMUART I/O Signal Descriptions	471
Table 465	Soft Reset Bit Definition for the MMUART_x	474
Table 466	MSS MMUART APIs	488
Table 467	MMUART Register Definitions	490

Table 468	RBR	491
Table 469	THR	491
Table 470	FCR	491
Table 471	DLR	493
Table 472	DMR	493
Table 473	DFR	493
Table 474	Baud Rates and Divisor Values for the 18.432 MHz Reference Clock	494
Table 475	IER	494
Table 476	IEM	494
Table 477	IIR	495
Table 478	Interrupt Identification Bit Values	495
Table 479	IIM	496
Table 480	LCR	496
Table 481	MCR	497
Table 482	LSR	498
Table 483	MSR	499
Table 484	SR	500
Table 485	MM0	500
Table 486	MM1	501
Table 487	MM2	501
Table 488	GFR	502
Table 489	TTG	503
Table 490	RTO	503
Table 491	ADR	503
Table 492	SPI Interface Signals	506
Table 493	Data Transfer Modes	507
Table 494	Summary of Master SPI Modes	507
Table 495	Behavior of the Output Enable Signal	510
Table 496	Soft Reset Bit Definitions for SPI Peripheral	517
Table 497	MSS SPI APIs	524
Table 498	SYSREG Control Registers	527
Table 499	SPI Register Summary	527
Table 500	CONTROL	528
Table 501	TXRXDF_SIZE	529
Table 502	Status	530
Table 503	INT_CLEAR	531
Table 504	RX_DATA	531
Table 505	TX_DATA	531
Table 506	CLK_GEN	531
Table 507	SLAVE_SELECT	532
Table 508	CLK_MODE Example, APB Clock = 153.8 MHz	532
Table 509	MIS	533
Table 510	RIS	533
Table 511	CONTROL2	534
Table 512	COMMAND	535
Table 513	PKTSIZE	536
Table 514	CMD_SIZE	536
Table 515	HWSTATUS	536
Table 516	STAT8	537
Table 517	I ² C Interface Signals	540
Table 518	Soft Reset Bit Definitions For I ² C Peripherals	541
Table 519	MSS I ² C APIs	546
Table 520	I ² C Register Map	550
Table 521	Control Register (CTRL)	551
Table 522	Clock Rate (CR)	551
Table 523	Status Register (STATUS)	552
Table 524	Status Register – Master-Transmitter Mode	552
Table 525	STATUS Register – Master-Receiver Mode	553
Table 526	STATUS Register – Slave-Receiver Mode	554

Table 527	STATUS Register – Slave-Transmitter Mode	556
Table 528	STATUS Register – Miscellaneous States	558
Table 529	Data Register (DATA)	558
Table 530	Slave0 Address Register (Slave0 ADR)	559
Table 531	SMBus Register (SMBUS)	559
Table 532	Frequency Register (FREQ)	560
Table 533	Glitch Register (GLITCHREG)	561
Table 534	Slave1 Address Register (SLAVE1 ADR)	561
Table 535	GPIO_X_CFG	565
Table 536	MSS GPIO Interrupts	566
Table 537	MSS GPIO APIs	573
Table 538	MSS GPIO Register Map	574
Table 539	GPIO SYREG Registers	576
Table 540	SOFT_RESET_CR	576
Table 541	MSS_GPIO_DEF	577
Table 542	LOOPBACK_CR	577
Table 543	GPIN_SRC_SEL_CR	577
Table 544	GPIO_SYSRESET_SEL_CR	578
Table 545	Associated IOMUXes for GPIOs	578
Table 546	IOMUX CELL 11	579
Table 547	IOMUX CELL 12	579
Table 548	IOMUX CELL 13	579
Table 549	IOMUX CELL 14	580
Table 550	IOMUX CELL 15	580
Table 551	IOMUX CELL 16	580
Table 552	IOMUX CELL 17	581
Table 553	IOMUX CELL 18	581
Table 554	IOMUX CELL 19	581
Table 555	IOMUX CELL 20	582
Table 556	IOMUX CELL 21	582
Table 557	IOMUX CELL 22	582
Table 558	IOMUX CELL 23	582
Table 559	IOMUX CELL 24	583
Table 560	IOMUX CELL 25	583
Table 561	IOMUX CELL 26	583
Table 562	IOMUX CELL 27	584
Table 563	IOMUX CELL 28	584
Table 564	IOMUX CELL 29	584
Table 565	IOMUX CELL 30	584
Table 566	IOMUX CELL 31	585
Table 567	IOMUX CELL 32	585
Table 568	IOMUX CELL 33	585
Table 569	IOMUX CELL 34	585
Table 570	IOMUX CELL 35	586
Table 571	IOMUX CELL 36	586
Table 572	IOMUX CELL 37	586
Table 573	IOMUX CELL 38	586
Table 574	IOMUX CELL 39	587
Table 575	IOMUX CELL 40	587
Table 576	IOMUX CELL 41	587
Table 577	IOMUX CELL 42	587
Table 578	IOMUX CELL 43	588
Table 579	IOMUX CELL 44	588
Table 580	IOMUX CELL 45	588
Table 581	IOMUX CELL 46	588
Table 582	IOMUX CELL 47	589
Table 583	IOMUX CELL 48	589
Table 584	IOMUX CELL 49	589
Table 585	IOMUX CELL 50	589

Table 586	IOMUX CELL 51	590
Table 587	IOMUX CELL 52	590
Table 588	IOMUX CELL 53	590
Table 589	IOMUX CELL 54	590
Table 590	IOMUX CELL 55	591
Table 591	IOMUX CELL 56	591
Table 592	APIs for the COMM_BLK	596
Table 593	COMM_BLK Register Map	597
Table 594	CONTROL	598
Table 595	STATUS	598
Table 596	INT_ENABLE	599
Table 597	DATA8	599
Table 598	DATA32	600
Table 599	FRAME_START8	600
Table 600	FRAME_START32	600
Table 601	Calendar Counter Description	602
Table 602	RTC Interface Signals	603
Table 603	RTC APIs	607
Table 604	Allocation of Bits in Calendar Mode	609
Table 605	Register Bit Allocation	609
Table 606	Register Map for RTC	609
Table 607	Control	610
Table 608	Mode	611
Table 609	Prescaler	611
Table 610	Alarm and Compare	612
Table 611	Date and Time	612
Table 612	The RTC_WAKEUP_CR in the SYSREG Block	613
Table 613	Timer Interface Signals	615
Table 614	Soft Reset Bit Definitions for System Peripheral	615
Table 615	MSS Timer APIs	619
Table 616	Timer Register Map	622
Table 617	TIMx_VAL	623
Table 618	TIMx_LOADVAL	623
Table 619	TIMx_BGLOADVAL	623
Table 620	TIMx_CTRL	624
Table 621	TIMx_RIS	624
Table 622	TIMx_MIS	625
Table 623	TIM64_VAL_U	625
Table 624	TIM64_VAL_L	625
Table 625	TIM64_LOADVAL_U	625
Table 626	TIM64_LOADVAL_L	626
Table 627	TIM64_BGLOADVAL_U	626
Table 628	TIM64_BGLOADVAL_L	627
Table 629	TIM64_CTRL	627
Table 630	TIM64_RIS	628
Table 631	TIM64_MIS	628
Table 632	TIM64_MODE	628
Table 633	Watchdog Timer Interface Signals	631
Table 634	Watchdog Timer APIs	636
Table 635	Watchdog Timer Register Interface Summary	638
Table 636	WDOGVALUE	638
Table 637	WDOGLOAD	638
Table 638	WDOGMVRP	639
Table 639	WDOGREFRESH	639
Table 640	WDOGENABLE	639
Table 641	WDOGCONTROL	640
Table 642	WDOGSTATUS	640
Table 643	WDOGRIS	640
Table 644	Watchdog Timer SYSREG	641

Table 645	VDD Power-Up to Functional Time	648
Table 646	DEVRST_N Power-Up to Functional Time	651
Table 647	GPIO_OUT Bank Reset Generation	659
Table 648	Switch Register Map	669
Table 649	Register Types	671
Table 650	SYSREG	676
Table 651	Subset of System Registers	682
Table 652	ESRAM_CR	683
Table 653	ESRAM_MAX_LAT	683
Table 654	eSRAM Maximum Latency Values	684
Table 655	DDR_CR	684
Table 656	ENVM_CR	684
Table 657	SW_ENVMREMAPSIZE	686
Table 658	ENVM_REMAP_BASE_CR	686
Table 659	ENVM_REMAP_FAB_CR	687
Table 660	CC_CR	687
Table 661	CC_REGION_CR	688
Table 662	CC_LOCK_BASE_ADDR_CR	688
Table 663	CC_FLUSH_INDX_CR	688
Table 664	DDRB_BUF_TIMER_CR	689
Table 665	DDRB_NB_ADDR_CR	689
Table 666	DDRB_NB_SIZE_CR	689
Table 667	Non-Bufferable Region	689
Table 668	DDRB_CR	690
Table 669	EDAC_CR	691
Table 670	MASTER_WEIGHT0_CR	691
Table 671	MASTER_WEIGHT1_CR	692
Table 672	Programmable Weight Values	692
Table 673	SOFT_IRQ_CR	693
Table 674	SOFT_RESET_CR	693
Table 675	M3_CR	695
Table 676	FAB_IF_CR	695
Table 677	LOOPBACK_CR	696
Table 678	GPIO_SYSRESET_SEL_CR	696
Table 679	GPIN_SRC_SEL_CR	697
Table 680	MDDR_CR	697
Table 681	USB_IO_INPUT_SEL_CR	698
Table 682	PERIPH_CLK_MUX_SEL_CR	698
Table 683	WDOG_CR	699
Table 684	MDDR_IO_CALIB_CR	699
Table 685	EDAC_IRQ_ENABLE_CR	699
Table 686	USB_CR	701
Table 687	ESRAM_PIPELINE_CR	701
Table 688	MSS_IRQ_ENABLE_CR	702
Table 689	RTC_WAKEUP_CR	702
Table 690	MAC_CR	702
Table 691	MSSDDR_PLL_STATUS_LOW_CR	703
Table 692	FACC_PLL_RANGE	704
Table 693	MSSDDR_PLL_STATUS_HIGH_CR	704
Table 694	MSSDDR_FACC1_CR	705
Table 695	Clock Ratio	707
Table 696	MSSDDR_FACC2_CR	707
Table 697	PLL_LOCK_EN_CR	709
Table 698	MSSDDR_CLK_CALIB_CR	709
Table 699	PLL_DELAY_LINE_SEL_CR	709
Table 700	MAC_STAT_CLRONRD_CR	710
Table 701	RESET_SOURCE_CR	710
Table 702	CC_DC_ERR_ADDR_SR	711
Table 703	CC_IC_ERR_ADDR_SR	711

Table 704	CC_SB_ERR_ADDR_SR	711
Table 705	CC_IC_MISS_CNTR_SR	711
Table 706	CC_IC_HIT_CNTR_SR	711
Table 707	CC_DC_MISS_CNTR_SR	712
Table 708	CC_DC_HIT_CNTR_CR	712
Table 709	CC_IC_TRANS_CNTR_SR	712
Table 710	CC_DC_TRANS_CNTR_SR	712
Table 711	DDRB_DS_ERR_ADR_SR	713
Table 712	DDRB_HPD_ERR_ADR_SR	713
Table 713	DDRB_SW_ERR_ADR_SR	713
Table 714	DDRB_BUF_EMPTY_SR	714
Table 715	DDRB_DSBL_DN_SR	714
Table 716	ESRAM0_EDAC_CNT	715
Table 717	ESRAM1_EDAC_CNT	715
Table 718	MAC_EDAC_TX_CNT	715
Table 719	MAC_EDAC_RX_CNT	715
Table 720	USB_EDAC_CNT	716
Table 721	CAN_EDAC_CNT	716
Table 722	ESRAM0_EDAC_ADR	716
Table 723	ESRAM1_EDAC_ADR	716
Table 724	MAC_EDAC_RX_ADR	717
Table 725	MAC_EDAC_TX_ADR	717
Table 726	CAN_EDAC_ADR	717
Table 727	USB_EDAC_ADR	717
Table 728	MM0_1_2_SECURITY	718
Table 729	MM4_5_DDR_FIC_SECURITY/MM4_5_FIC64_SECURITY	718
Table 730	MM3_6_7_8_SECURITY	719
Table 731	MM9_SECURITY	720
Table 732	M3_SR	721
Table 733	ETM_COUNT_LOW	721
Table 734	ETM_COUNT_HIGH	721
Table 735	DEVICE_SR	722
Table 736	ENVM_PROTECT_USER	722
Table 737	ENVM_STATUS	724
Table 738	DEVICE_VERSION	724
Table 739	MSSDDR_PLL_STATUS	724
Table 740	USB_SR	725
Table 741	ENVM_SR	725
Table 742	DDRB_STATUS	726
Table 743	MDDR_IO_CALIB_STATUS	726
Table 744	MSSDDR_CLK_CALIB_STATUS	727
Table 745	WDOGLOAD	727
Table 746	WDOGMVRP	727
Table 747	USERCONFIG0	727
Table 748	USERCONFIG1	727
Table 749	USERCONFIG2	728
Table 750	USERCONFIG3	728
Table 751	FAB_PROT_SIZE	728
Table 752	Region Size	728
Table 753	FAB_PROT_BASE	729
Table 754	MSS_GPIO_DEF	730
Table 755	EDAC_SR	730
Table 756	MSS_INTERNAL_SR	731
Table 757	MSS_EXTERNAL_SR	731
Table 758	WDOGTIMEOUTEVENT	733
Table 759	CLR_MSS_COUNTERS	733
Table 760	CLR_EDAC_COUNTERS	733
Table 761	FLUSH_CR	734
Table 762	MAC_STAT_CLR_CR	735

Table 763	IOMUXCELL_CONFIG[n]	736
Table 764	MSS_IOMUXSEL5 [N][2:0]	737
Table 765	MSS_IOMUXSEL4[N][2:0]	737
Table 766	Interrupt Line Signal Distribution	740
Table 767	FIIC Port List	741
Table 768	Interrupt Source Numbers	744
Table 769	Bit-Band Register Bit of Interrupt_Enable0 and Interrupt_Enable1	745
Table 770	Bit-Band Register Bit of INTERRUPT_REASON0 and INTERRUPT_REASON1	747
Table 771	SmartFusion2 SoC FPGA FIIC Register Map	749
Table 772	INTERRUPT_ENABLE0	749
Table 773	INTERRUPT_ENABLE1	753
Table 774	INTERRUPT_REASON1	753
Table 775	INTERRUPT_REASON0	754
Table 776	INTERRUPT_MODE	756
Table 777	Number of FICs Available for Use in Each Device	758
Table 778	Master Group Access to Fabric Slaves	760
Table 779	FIC Memory Regions	760
Table 780	Fabric Interface Controller Port List	762
Table 781	Address Regions and Compatible Slots for 256 MB Per Slot Option	770
Table 782	FAB_IF Register in the SYSREG Block	783
Table 783	FDDR APB Slave Configuration Interface Port List	785
Table 784	MDDR APB Slave Configuration Interface Port List	785
Table 785	SERDERIF APB Slave Configuration Interface Port List	786
Table 786	MSS APB Master Configuration Interface Port List	786
Table 787	Minimum Number of Checksum Bits Required Data	793

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 15.0

The following changes were made in revision 15.0 of this document.

- [Configuration Through Libero Software and Firmware](#), page 16 updated.
- TESMAC Firmware Drivers information updated in [Table 327](#), page 391, [Table 328](#), page 391, [Table 329](#), page 391, [Table 330](#), page 392.
- Updated register information in [Table 468](#), page 491, [Table 666](#), page 689, [Table 693](#), page 704, [Table 722](#), page 716, [Table 723](#), page 716, [Table 724](#), page 717, [Table 725](#), page 717, [Table 726](#), page 717, and [Table 727](#), page 717.
- Updated [ULPI \(UTMI+ Low Pin Interface\) I/O Interface](#), page 287.

1.2 Revision 14.0

The following changes were made in revision 14.0 of this document.

- The power-up to functional time sequence information was updated to include the POWER_ON_RESET_N signal. For more information, see [Power-Up to Functional Time Sequence](#), page 645.
- The test cases listed in [Table 645](#), page 648 were updated for consistency with [DS0128: IGLOO2 and SmartFusion2 Datasheet](#). The VDD power-up to functional time flow diagram ([Figure 282](#), page 650) was updated to include the POWER_ON_RESET_N signal. For more information, see [VDD Power-Up to Functional Time](#), page 648.
- The test cases listed in [Table 646](#), page 651 were updated for consistency with [DS0128: IGLOO2 and SmartFusion2 Datasheet](#). The DEVRST_N power-up to functional time flow diagram ([Figure 283](#), page 651) was updated to include the POWER_ON_RESET_N signal. For more information, see [DEVRST_N Power-Up to Functional Time](#), page 651.

1.3 Revision 13.0

The following changes were made in revision 13.0 of this document.

- Updated [Power Management](#), page 45 (SAR 84698).
- Updated [Table 72](#), page 120 (SAR 83563).
- Added [System Registers Behavior for M2S005/010 Devices](#), page 682 (SAR 83541).
- Updated [Table 738](#), page 724 (SAR 83014).
- Updated [Software Reset Control Register](#), page 693 (SAR 82129).
- Updated [Figure 82](#), page 160, [Figure 83](#), page 160, [Figure 84](#), page 161, [Figure 85](#), page 161, and [Figure 86](#), page 162 (SAR 85251).
- Updated [Table 113](#), page 184 (SAR 85241).

1.4 Revision 12.0

The following changes were made in revision 12.0 of this document.

- Updated [Table 650](#), page 676 (SAR 70453).
- Updated bit name from MSS_IOMUXSEL4UPPER[N] to MSS_IOMUXSEL4[N][2:0] in the [Table 763](#), page 736 (SAR 78918).
- Added [Register Lock Bits Configuration](#), page 674 (SAR 79854).
- Updated [Table 777](#), page 758 (SAR 78910).
- Updated [Power-On Reset Generation Sequence](#), page 643 (SAR 76102).
- Updated Bit number [18:1] description in [Table 658](#), page 686 (SAR 69988).
- Updated Register name CLRHINT[2:0] row in [Table 112](#), page 180 (SAR 74635).
- Updated \leq to \leq in [Cortex-M3 Processor Overview and Debug Features](#), page 6 (SAR 76277).
- Added [Figure 85](#), page 161 (M2S060 device) (SAR 78896).

- Added clocking information to [SGMII Module](#), page 377 (SAR 75342).
- Added information regarding full behavioral simulation model in the applicable chapters (SAR 80669).
- Removed note from [Peripheral Signals Assignment Table](#), page 443, and removed “System Clock Frequency” section (SAR 53174).
- Added [Power-Up to Functional Time Data](#), page 646 (SAR 81600).
- Added [Error Detection and Correction Controllers](#), page 792 (SAR 80945).
- Updated [Figure 153](#), page 375, [Figure 155](#), page 380, [Figure 160](#), page 387, and [Figure 161](#), page 388 (SAR 81447).
- Added a note for [Functional Description](#), page 7 in [Cortex-M3 Processor Overview and Debug Features](#), page 6 (SAR 81261).
- Updated [Table 640](#), page 639 and [Table 641](#), page 640 (SAR 82053).
- Updated Bit number 2 description in [Table 693](#), page 704 (SAR 82131).

1.5 Revision 11.0

The following changes were made in revision 11.0 of this document.

- Updated [Table 367](#), page 413 and [Table 368](#), page 413 (SAR 58939).
- Updated [Locked Transactions](#), page 231 (SAR 64773).

1.6 Revision 10.0

The following changes were made in revision 10.0 of this document.

- Updated [Power-Up to Functional Time Sequence](#), page 645 (SAR 72958).
- Updated [Table 481](#), page 497 (SAR 70358).
- Updated [Functional Description](#), page 146 chapter in [Embedded NVM \(eNVM\) Controllers](#), page 145 chapter (SAR 73736).
- Updated [Table 741](#), page 725 and [Table 110](#), page 180 (SAR 70182).
- Added [Figure 82](#), page 160, [Figure 83](#), page 160, [Figure 84](#), page 161, and [Figure 86](#), page 162, and added [eNVM Pages for Special Purpose Storage](#), page 163 (SAR 66208).
- Updated [SPI Use Models](#), page 525 (SAR 62152).
- Updated [Table 104](#), page 175, [Table 112](#), page 180, and [Table 656](#), page 684 (SAR 71776).

1.7 Revision 9.0

The following changes were made in revision 9.0 of this document.

- Updated [Reset Controller](#), page 642 (SAR 51042, 65634, 66764, 66981).
- Updated [Figure 296](#), page 661 (SAR 67010)
- Added a note to [System Reset](#), page 654 (SAR 64029)
- Updated [FIC Implementation Considerations](#), page 766 (SAR 64802).

1.8 Revision 8.0

The following changes were made in revision 8.0 of this document.

- Removed M2S100 devices list from [Features](#), page 145, [Table 93](#), page 146, [Table 94](#), page 147, and [Table 114](#), page 185 (SAR 62858).
- Updated [Table 104](#), page 175 for NV_FREQRNG and replaced FREQRNG with NV_FREQRNG throughout the document (SAR 62544)
- Updated [Page Program](#), page 152 (SAR 61046).
- Updated [System Register Block](#), page 670 (SAR 62544).
- Added a reference to *SmartFusion2 SoC FPGA High Speed DDR Interfaces User Guide* in the [HPDMA Use Models](#), page 245 (SAR 60106).

1.9 Revision 7.0

The following changes were made in revision 3.0 of this document.

- Updated [Cache Engine](#), page 139 (SAR 58874).

1.10 Revision 6.0

The following changes were made in revision 3.0 of this document.

- Added a note to [Trace Port Interface Unit \(TPIU\) Configuration](#), page 17 (SAR 55243).
- Updated [Table 3](#), page 14 and [Table 4](#), page 15 (SAR 51221).
- Updated [Table 4](#), page 15 (SAR 57010).
- Updated [Memory System Ordering of Memory Accesses](#), page 30 (SAR 49728).
- Updated [Power Management](#), page 45 (SAR 50897).
- Changed S bus to SBUS in [Cache Controller](#), page 133 (SAR 48038).
- Added a note to [Cache Locked Mode](#), page 141 (SAR 46463).
- Added notes in [How to Use eNVM](#), page 165 (SARs 47902, 46463, 55243, and 49367).
- Updated [Set Lock Bit and User Unlock Commands](#), page 153 (SAR 55728).
- Added a note to [Page Program](#), page 152 (SAR 56496).
- Updated the document (SARs 46656, 57682, 55965, and 52508).
- Added [eNVM Program and Verify Operations Timing Diagrams](#), page 155 (SAR 44080).
- Updated the HPDMA [Details of Operation](#), page 241 (SARs 48855 and 46151).
- Updated [Posted APB Writes](#), page 270 (SAR 48614).
- Added a note to [PHY Interfaces](#), page 287 and to [ULPI \(UTMI+ Low Pin Interface\) I/O Interface](#), page 287 (SAR 50776).
- Updated [Table 323](#), page 381 (SAR 51248).
- Updated [Table 321](#), page 378 (SAR 52741).
- Added [CoreMACFilter Overview](#), page 434 (SAR 58615).
- Updated [Table 500](#), page 528 for TXRXDFCOUNT value (SAR 48912).
- Updated [MSS GPIO Functional Description](#), page 563 (SAR 41775).
- Added a note to [Table 543](#), page 577 (SAR 51119).
- Added a note to [Power-On Reset Generation Sequence](#), page 643 (SAR 56613).
- Updated [Table 763](#), page 736 for bit numbers (SAR 52993).
- Updated [Table 772](#), page 749 (SAR 50361).
- Updated the introductory content of [Fabric Interface Controller](#), page 757, and added a note to [Figure 336](#), page 766 (SAR 56584).
- Updated [Embedded NVM \(eNVM\) Controllers](#), page 145, [Embedded SRAM \(eSRAM\) Controllers](#), page 187, [AHB Bus Matrix](#), page 210, [High Performance DMA Controller](#), page 236, [Peripheral DMA](#), page 264, [Serial Peripheral Interface Controller](#), page 504, [Communication Block](#), page 592, [System Register Block](#), page 670, [Fabric Interface Interrupt Controller](#), page 738, and [Fabric Interface Controller](#), page 757 chapters for FTC comments (SAR 56085).

1.11 Revision 5.0

The following changes were made in revision 3.0 of this document.

- Updated the document (SAR 53559).
- Updated [Figure 289](#), page 657 and [Figure 290](#), page 657 (SAR 53672).
- Updated [Figure 356](#), page 782 (SAR 54025).

1.12 Revision 4.0

The following changes were made in revision 4.0 of this document.

- Corrected all instances of baud rate (SAR 47848).
- References to 1.0 v were removed (SAR 46823).
- Updated [Features](#), page 133 of [Cache Controller](#), page 133 (SAR 42397).
- Added a reference to the *Remapping eNVM, eSRAM, and DDR/SDR SDRAM Memories application note* (SAR 47903).
- Updated [Figure 57](#), page 134 (SAR 48354).
- Updated [eNVM Theory of Operation](#), page 148 (SARs 47905 and 50072).
- Updated [Figure 112](#), page 225, [Figure 113](#), page 226, and [Figure 117](#), page 230, [Table 142](#), page 220, and [Table 146](#), page 232 (SARs 39867 and 48605).
- Updated the direction for ULPI_XCLK in [Table 190](#), page 287 (SAR 48602).
- Updated [Table 344](#), page 400 (SAR 46933).
- Updated [Figure 239](#), page 562 (SAR 48715).

- Updated [Table 611](#), page 612 (SAR 42472).
- Updated [Power-On Reset](#), page 654 (SAR 45905).
- Updated [Table 650](#), page 676, [Table 693](#), page 704, and [Table 715](#), page 714 (SAR 48598).
- Updated details for CC_EDAC_EN in [Table 669](#), page 691 (SAR 48194).
- Updated [Table 735](#), page 722 (SAR 44275 and 48634).
- Updated [Table 763](#), page 736 (SAR 49501).
- Updated details for CC_EDAC_EN in [Table 138](#), page 208 (SAR 48194)

1.13 Revision 3.0

The following changes were made in revision 3.0 of this document.

- Updated "Purpose" section (SAR 48036).
- Updated [Table 321](#), page 378 (SAR 47908).
- Updated links in [CAN Controller](#), page 436 (SAR 47608).
- Restructured [Fabric Interface Controller](#), page 757 as per inputs (SAR 47958).

1.14 Revision 2.0

The following changes were made in revision 2.0 of this document.

- Modified the title of the user guide (SAR 45509).
- Restructured [Cortex-M3 Processor Overview and Debug Features](#), page 6 (SAR 44811).
- Restructured [Cache Controller](#), page 133 (SAR 44811).
- Restructured [Embedded NVM \(eNVM\) Controllers](#), page 145 and updated [AHBL Address Map to NVM](#), page 150 (SAR 40367).
- Restructured [Embedded SRAM \(eSRAM\) Controllers](#), page 187 (SAR 45900).
- Restructured [AHB Bus Matrix](#), page 210 (SAR 46456).
- Restructured [High Performance DMA Controller](#), page 236 (SAR 46151).
- Restructured [Peripheral DMA](#), page 264 (SAR 46422).
- Restructured [Universal Serial Bus OTG Controller](#), page 284 (SAR 47042).
- Restructured [Ethernet MAC](#), page 374 (SAR 47043).
- Restructured [CAN Controller](#), page 436 (SAR 50262).
- Restructured [MMUART Peripherals](#), page 469 (SAR 50262).
- Restructured [Serial Peripheral Interface Controller](#), page 504 (SAR 50262).
- Restructured [Inter-Integrated Circuit Peripherals](#), page 538 (SAR 50262).
- Restructured [MSS GPIO](#), page 562 (SAR 50262).
- Restructured [Communication Block](#), page 592 (SAR 50262).
- Restructured [RTC System](#), page 601 (SAR 46060).
- Restructured [System Timer](#), page 614 (SAR 46048).
- Restructured [Watchdog Timer](#), page 629 (SAR 46053).
- Restructured [Reset Controller](#), page 642 and updated [Power-Up to Functional Time Sequence](#), page 645 (SAR 42469).
- Updated [Figure 294](#), page 660 (SAR 43874).
- Restructured [System Register Block](#), page 670 (SAR 47001).
- Updated the "Flash Write ProtectSYSREG Block Register Write Protection" section (SAR 41978).
- Updated [Table 657](#), page 686, [Table 739](#), page 724, [Table 764](#), page 737, and [Table 765](#), page 737 (SARs 43008 and 42733).
- Restructured [Fabric Interface Interrupt Controller](#), page 738 (SAR 50262).
- Restructured [Fabric Interface Controller](#), page 757 (SAR 45631).
- Restructured [APB Configuration Interface](#), page 784 (SAR 50262).
- Updated [DDR Remap](#), page 229 (SAR 42910).
- Updated [Table 779](#), page 760 (SAR 45578).

1.15 Revision 1.0

Revision 1.0 was the first publication of this document.

- Updated [Cache Controller](#), page 133 (SAR 41865).
- [Table 89](#), page 135, [Figure 57](#), page 134, and [Table 91](#), page 137 (SAR 41865).
- Added [Figure 250](#), page 593 (SAR 41229).

- Added [Figure 328](#), page 761 through [Figure 336](#), page 766 (SAR 39058).

2 Cortex-M3 Processor Overview and Debug Features

The ARM Cortex-M3 processor is a low power consumption processor that features low gate count, low interrupt latency, and low-cost debug. It is intended for deeply embedded applications that require optimal interrupt response features. This processor implements the ARM v7-M architecture and is shown in [Figure 1](#), page 7. The SmartFusion[®]2 SoC FPGA device uses the R2P1 version of the Cortex-M3 core. This chapter highlights the Cortex-M3 processor and debug subsystem customizations made specific to SmartFusion2.

For more details on the internals like programming model, exception model, instruction set, the Cortex-M3 specific peripherals such as SysTick timer, memory protection unit and others, refer to the [Cortex-M3 Processor \(Reference Material\)](#), page 18. The following manuals are available at the [ARM Info center](#):

- [Cortex-M3 Technical Reference Manual](#)
- [ARM v7-M Architecture Reference Manual](#)
- [ARM v7-M Architecture Application Level Reference Manual](#)

The *Definitive Guide to the ARM Cortex-M3* by Joseph Yiu is recommended as additional reading (ISBN: 978-0-7506-8534-4).

2.1 Features

- A 32-bit processor core with low gate count and low latency interrupt processing.
- A RISC processor, with 3-stage pipeline Harvard architecture, pipeline core incorporating branch speculation, single cycle multiplication, and hardware division, giving a Dhrystone benchmark of 1.25 DMIPS/MHz.
- A nested vectored interrupt controller (NVIC) that closely integrates with the processor core to achieve low latency interrupt processing.
- A memory protection unit (MPU) is included. This facilitates the protected memory regions creation and setting access rights for the protected regions.
- A Cortex-M3 processor, which is configured for SmartFusion2 MSS, and uses only little-endian.
- An auxiliary control register is included.
- Multiple high-performance bus interfaces that are connected through an advanced high-performance bus (AHB).
- A debug solution with the optional ability to:
 - Implement breakpoints and code patches
 - Implement watchpoints, tracing, and system profiling
 - Support print style debugging
 - Bridge to a trace port analyzer

Manufacturers of Cortex-M3 processor integrated circuits are permitted some latitude in configuring a particular implementation of the Cortex-M3 processor delivered by ARM. The following features are implementation specifics in the SmartFusion2 device:

- MPU: This helps in creating protected and protected regions of memory
- Flash patch break point (FPB)
- Data watchpoint and trace (DWT) unit
- Instrumental trace macrocell (ITM)
- Embedded trace macrocell (ETM)
- Power-mode saving:
 - HCLK is gated off when in SLEEPING or SLEEPDEEP mode.
 SLEEPING and SLEEPDEEP signals are available at the FPGA fabric interface sleep mode extension handshake signals are available at the FPGA fabric interface.

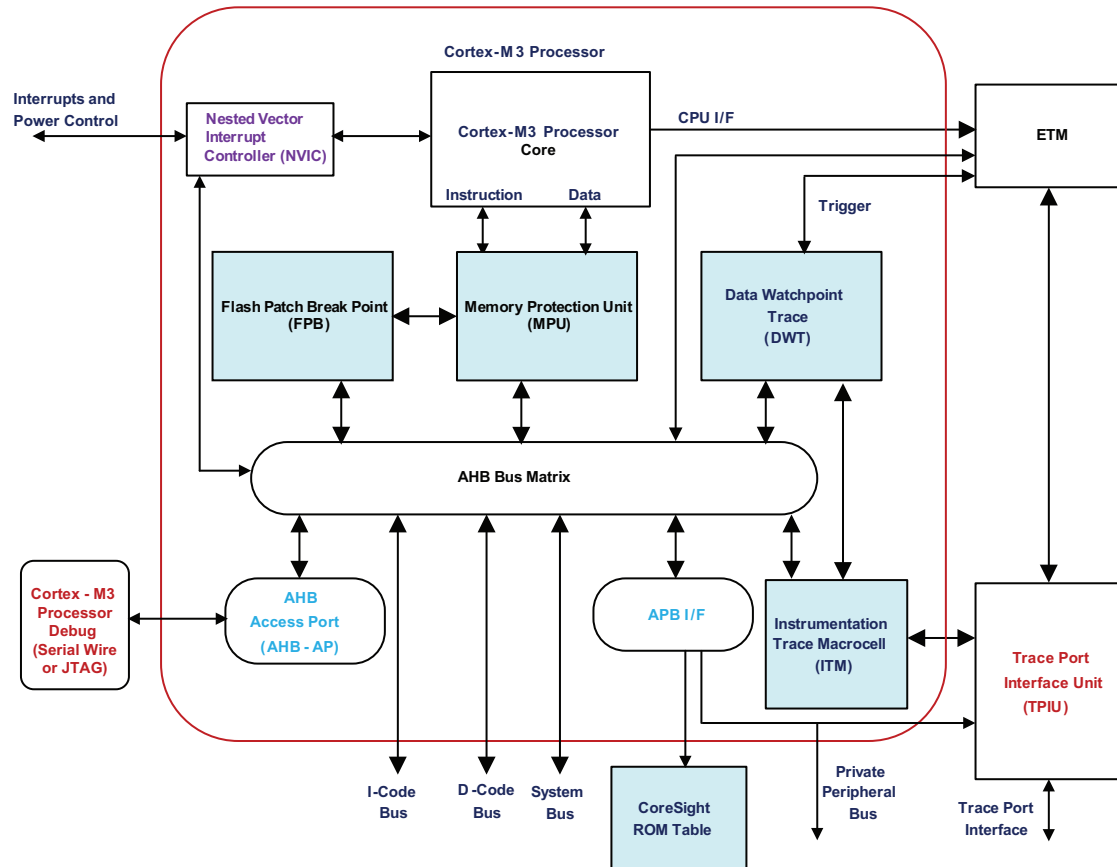
- Not all registers in the register bank are reset
- Endianness: little endian only
- Auxiliary control register is included
- Wake-up interrupt controller (WIC) is not included

For more details of these configurations and optional features, refer to [Cortex-M3 Processor \(Reference Material\)](#), page 18.

2.2 Functional Description

The following figure shows the Cortex-M3 processor, core peripherals and debug subsystem implementations used in SmartFusion2.

Figure 1 • Cortex-M3 Processor R2P1 Block Diagram as Implemented in the SmartFusion2 SoC FPGA



The following topics are covered in detail in the sub-sections:

- Cortex-M3 Processor NVIC
- Cortex-M3 Processor SysTick Timer
- Cortex-M3 Processor Debug Subsystem
- Data Watch Point (DWP) and Trace
- Instrumentation Trace Macrocell
- Embedded Trace Macrocell

Note: The Cortex-M3 operating frequency is dependent on device speed grade (up to 166 MHz). Refer to SmartFusion2 Specifications-MSS Clock Frequency section from [DS0128: IGL002 FPGA and SmartFusion2 SoC FPGA Datasheet](#) for more information.

2.3 Cortex-M3 Processor NVIC

The Cortex-M3 processor contains an NVIC, which is responsible for:

- Facilitating low-latency exception and interrupt handling
- Controlling power management

The NVIC supports 11 exceptions as shown in [Table 1](#), page 8. The NVIC also supports up to 83 dynamically re-prioritizable external interrupts, as shown in the [Table 2](#), page 8, each with up to 16 levels of priority. The NVIC maintains knowledge of stacked (nested) interrupts to enable tail-chaining of interrupts. In MSS, the NVIC is configured to have 16 levels of priority (4 msb in BASEPRI register) are implemented, so BASEPRI register [7-4] are used for the priority setting and [3-0] are read as zeros.

The following table lists exceptions. The detailed description of these exceptions can be found in the [ARM Cortex-M3 Technical Reference Manual](#).

Table 1 • Cortex-M3 Processor Exceptions

Cortex-M3 Exceptions	Position in Interrupt Vector Table	Priority	Description
Reset	1 (zero position is stack pointer)	–3	Invoked on power-up and reset
Non-maskable exception	2	–2	Non-maskable interrupt (NMI)—watchdog timeout interrupt
HardFault	3	–1	Hard fault interrupt: all fault conditions if the corresponding fault handler is not enabled
Memory management exception	4	Configurable	Memory management interrupt: memory management fault; MPU violation or access to illegal locations.
Bus fault exception	5	Configurable	Bus fault interrupt: bus error; occurs when the AHB interface receives an error response from a bus slave (also called prefetch abort if it is an instruction fetch or data abort if it is a data access).
UsageFault	6	Configurable	Usage fault interrupt: exceptions resulting from a program error or trying to access a coprocessor (the Cortex-M3 does not support a coprocessor).
SVCcall	11	Configurable	Supervisory call interrupt
Debug monitor	12	Configurable	Debug monitor interrupt: breakpoints, watchpoints, or external debug requests
PendSV	14	Configurable	Pend supervisory interrupt
SysTick	15	Configurable	System tick timer interrupt

The interrupt sources listed in the following table are connected to the NVIC of the Cortex-M3 processor in the MSS.

Table 2 • Cortex-M3 Processor Interrupts

Cortex-M3 Interrupt	Signal	Source	Description
INTNMI	WDGTIMEOUTINT	WATCHDOG	This interrupt is asserted (if enabled) if the counter reaches zero and interrupt rather than reset generation has been selected on counter timeout.
INTISR[0]	WDGWAKEUPINT	WATCHDOG	This interrupt is asserted (if enabled) on crossing the WDOGMVRP level when the SLEEPING input is asserted.

Table 2 • Cortex-M3 Processor Interrupts (continued)

Cortex-M3 Interrupt	Signal	Source	Description
INTISR[1]	RTC_WAKEUP_INTR	RTC	RTC match/wake up interrupt from RTC block
INTISR[2]	SPIINT0	SPI_0	Interrupt from SPI 0
INTISR[3]	SPIINT1	SPI_1	Interrupt from SPI 1
INTISR[4]	I2C_INT0	I2C_0	Interrupt from I2C 0
INTISR[5]	I2C_SMBALERT0	I2C_0	Interrupt from I2C 0
INTISR[6]	I2C_SMBUS0	I2C_0	Interrupt from I2C 0
INTISR[7]	I2C_INT1	I2C_1	Interrupt from I2C 1
INTISR[8]	I2C_SMBALERT1	I2C_1	Interrupt from I2C 1
INTISR[9]	I2C_SMBUS1	I2C_1	Interrupt from I2C 1
INTISR[10]	MMUART0_INTR	MMUART_0	Interrupt from MMUART 0
INTISR[11]	MMUART1_INTR	MMUART_1	Interrupt from MMUART 1
INTISR[12]	MAC_INT	MAC	Interrupt from Ethernet MAC
INTISR[13]	PDMAINTERRUPT	PDMA	Interrupt from peripheral DMA block
INTISR[14]	TIMER1_INTR	TIMER	Timer1 interrupt
INTISR[15]	TIMER2_INTR	TIMER	Timer2 interrupt
INTISR[16]	CAN_INTR	CAN	Interrupt from CAN
INTISR[17]	ENV_M_INT0	ENV_MTOAHB0	Asserted on an eNVM_0 basis at the completion of ERASE_PAGE, PROGRAM, etc.
INTISR[18]	ENV_M_INT1	ENV_MTOAHB1	Asserted on an eNVM_1 basis at the completion of ERASE_PAGE, PROGRAM, ECC ERROR, etc.
INTISR[19]	COMM_BLK_INTR	COMBLK	Communication block interrupt
INTISR[20]	USB_MC_INT	USB	CPU interrupts
INTISR[21]	USB_DMA_INT	USB	Core's DMA engine performs data transfer between endpoint memories and system memory via AHB master port. DMA controller-interrupt.
INTISR[22]	MSSDDR_PLL_LOCK_INT	SYSREG	Interrupt indicating that MSSDDR PLL has achieved lock.
INTISR[23]	MSSDDR_PLL_LOCKLOST_INT	SYSREG	Interrupt indicating that MSSDDR PLL has lost lock.
INTISR[24]	SW_ERRORINTERRUPT	SYSREG	If set, it indicates to the Cortex-M3 processor that: <ul style="list-style-type: none"> – One of the masters of the switch attempted an access that resulted in either an error termination by the slave (or possibly the switch itself) or – Was decoded as an access to unimplemented address space or o. If the master attempted an access while disabled or – In the case of the fabric master, attempted to access the protected region of memory space This signal is set by ORing the fields of SW_ERRORSTATUS. It is cleared by writing 1 to the SW_CLEARSTATUS bit.

Table 2 • Cortex-M3 Processor Interrupts (continued)

Cortex-M3 Interrupt	Signal	Source	Description
INTISR[25]	CACHE_ERRINTR	SYSREG	If asserted, indicates that the interrupt is coming from CACHE. This interrupt is generated in the SysReg by ORing of the various interrupts from the CACHE block: CC_HRESPERRINT0, CC_HRESPERRINT1, CC_HRESPERRINT2, CC_HRESPERRINT3.
INTISR[26]	DDRB_INTR	SYSREG	If asserted, indicates that the interrupt is coming from DDRBRDIGE module. Interrupts from MSS DDR Bridge module: DDRB_ERROR and DDRB_LOCKTIMEOUT. These interrupts are ORed in the SysReg and fed to the Cortex-M3 processor.
INTISR[27]	HPD_XFR_CMP_INT	HPDMA	It is asserted when any HPDMA completes a descriptor transfer. Once asserted, it remains asserted until cleared by means of writing 1 to the bit in the control register of the Descriptor-N (0, 1, 2, 3). If HPDMA completes more than one descriptor transfers before the interrupt is serviced then this bit remains asserted until all the descriptors have had Clr_D<N>_Xfr_cmp_int written to 1.
INTISR[28]	HPD_XFR_ERR_INT	HPDMA	It is asserted when any HPDMA completes a descriptor transfer with error. Once asserted, it remains asserted until cleared by means of writing 1 to the bit in the control register of the Descriptor-N (0, 1, 2, 3). If HPDMA completes more than one descriptor with errors before the interrupt is serviced then this bit remains asserted until all the descriptors have had Clr_D<N>_Xfr_err_int written to 1.
INTISR[29]	ECCINTR	SYSREG	It is asserted when an ECC error has been detected in ESRAM0, ESRAM1, CACHE, MAC, CAN, MDDR, and USB. This is generated by ORing ECC interrupts from these modules.
INTISR[30]	MDDR_IO_CALIB_INT	SYSREG	The interrupt is generated when MDDR calibration is finished. For the calibration after reset, this would be followed by locking the codes directly. However, for in-between runs during functional DDR operation, the assertion of interrupt does not guarantee lock as the state machine would wait for the ideal time (DRAM self-refresh) for locking. This can be used by the firmware to insert an ideal time, and provides an indication of availability of locked codes.
INTISR[31]	FAB_PLL_LOCK_INT	SYSREG	Interrupt indicating that MSSDDR PLL has achieved lock
INTISR[32]	FAB_PLL_LOCKLOST_INT	SYSREG	Interrupt indicating that MSSDDR PLL has lost lock

Table 2 • Cortex-M3 Processor Interrupts (continued)

Cortex-M3 Interrupt	Signal	Source	Description
INTISR[33]	FIC64_INT	SYSREG	This interrupt will be generated by FIC64 when one of the following conditions is true: Write error for HPDMA or switch WCBs (from DDR_AXI_INTF) Simultaneous read and write accesses by HPDMA and switch for same address Lock time out condition
INTISR[34]	F2H_INTERRUPT[0]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[35]	F2H_INTERRUPT[1]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[36]	F2H_INTERRUPT[2]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[37]	F2H_INTERRUPT[3]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[38]	F2H_INTERRUPT[4]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[39]	F2H_INTERRUPT[5]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[40]	F2H_INTERRUPT[6]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[41]	F2H_INTERRUPT[7]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[42]	F2H_INTERRUPT[8]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[43]	F2H_INTERRUPT[9]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[44]	F2H_INTERRUPT[10]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[45]	F2H_INTERRUPT[11]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[46]	F2H_INTERRUPT[12]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[47]	F2H_INTERRUPT[13]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[48]	F2H_INTERRUPT[14]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[49]	F2H_INTERRUPT[15]	FPGA fabric	Interrupt from the FPGA fabric
INTISR[50]	GPIO_INT[0]	GPIO	Interrupt from GPIO
INTISR[51]	GPIO_INT[1]	GPIO	Interrupt from GPIO
INTISR[52]	GPIO_INT[2]	GPIO	Interrupt from GPIO
INTISR[53]	GPIO_INT[3]	GPIO	Interrupt from GPIO
INTISR[54]	GPIO_INT[4]	GPIO	Interrupt from GPIO
INTISR[55]	GPIO_INT[5]	GPIO	Interrupt from GPIO
INTISR[56]	GPIO_INT[6]	GPIO	Interrupt from GPIO
INTISR[57]	GPIO_INT[7]	GPIO	Interrupt from GPIO
INTISR[58]	GPIO_INT[8]	GPIO	Interrupt from GPIO
INTISR[59]	GPIO_INT[9]	GPIO	Interrupt from GPIO
INTISR[60]	GPIO_INT[10]	GPIO	Interrupt from GPIO
INTISR[61]	GPIO_INT[11]	GPIO	Interrupt from GPIO
INTISR[62]	GPIO_INT[12]	GPIO	Interrupt from GPIO
INTISR[63]	GPIO_INT[13]	GPIO	Interrupt from GPIO
INTISR[64]	GPIO_INT[14]	GPIO	Interrupt from GPIO
INTISR[65]	GPIO_INT[15]	GPIO	Interrupt from GPIO

Table 2 • Cortex-M3 Processor Interrupts (continued)

Cortex-M3 Interrupt	Signal	Source	Description
INTISR[66]	GPIO_INT[16]	GPIO	Interrupt from GPIO
INTISR[67]	GPIO_INT[17]	GPIO	Interrupt from GPIO
INTISR[68]	GPIO_INT[18]	GPIO	Interrupt from GPIO
INTISR[69]	GPIO_INT[19]	GPIO	Interrupt from GPIO
INTISR[70]	GPIO_INT[20]	GPIO	Interrupt from GPIO
INTISR[71]	GPIO_INT[21]	GPIO	Interrupt from GPIO
INTISR[72]	GPIO_INT[22]	GPIO	Interrupt from GPIO
INTISR[73]	GPIO_INT[23]	GPIO	Interrupt from GPIO
INTISR[74]	GPIO_INT[24]	GPIO	Interrupt from GPIO
INTISR[75]	GPIO_INT[25]	GPIO	Interrupt from GPIO
INTISR[76]	GPIO_INT[26]	GPIO	Interrupt from GPIO
INTISR[77]	GPIO_INT[27]	GPIO	Interrupt from GPIO
INTISR[78]	GPIO_INT[28]	GPIO	Interrupt from GPIO
INTISR[79]	GPIO_INT[29]	GPIO	Interrupt from GPIO
INTISR[80]	GPIO_INT[30]	GPIO	Interrupt from GPIO
INTISR[81]	GPIO_INT[31]	GPIO	Interrupt from GPIO

2.4 Cortex-M3 Processor SysTick Timer

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads, that is, wraps to the value in the SYST_RVR register on the next clock edge, and then counts down on subsequent clocks. The SysTick timer is used to generate a periodic interrupt to the Cortex-M3 processor. The SysTick can be polled by the software or it can be configured to generate an interrupt. The SysTick interrupt has its own entry in the vector table and therefore its own handler.

2.5 Cortex-M3 Processor Debug Subsystem

2.5.1 Cortex-M3 Processor Debug Port

The debug port uses a serial wire (SW) JTAG debug port (SWJ-DP). This enables either the JTAG or the SW protocol to be used for debugging. The SWJ-DP defaults to JTAG mode at power-up and can be switched to SW by applying a specific sequence to the debug pins.

The trace port interface unit (TPIU) is configured to support ITM debug trace and ETM debug trace. Serial wire mode is used for the TPIU output data and this is overlaid on the JTAG TDO port. One implication of this is that instrumentation trace cannot be used along with JTAG-based debugging. SW debugging and ITM can be used together.

The Cortex-M3 processor provides the following debug Interfaces:

- SWJ-DP: JTAG is the industry-standard interface used to download and debug programs on a target processor, as well as for other functions. It offers access to all of the Cortex-M3 processor CoreSight® debug capabilities.
- SW-DP: The serial wire debug (SWD) mode is an alternative to the standard JTAG interface. SWD uses two pins to provide the same debug functionality as JTAG with no performance penalty, and introduces data trace capabilities with the serial wire viewer (SWV). The SWD interface pins are overlaid with the JTAG signals, allowing standard target connectors to be used.
 - TCLK: SWCLK (serial wire clock)
 - TMS: SWDIO (serial wire debug data input/output)

- DO: SWO (output pin for SWV, refer to the next section).
- SWV: It provides real-time data trace information from various sources within the Cortex-M3 processor device. This is output via the single serial wire output (SWO) pin while your system processor continues running at full speed. SWV can only be used with the SWD interface.
- ETM: The embedded trace macrocell provides high bandwidth instruction trace via four dedicated trace pins.

2.5.2 Cortex-M3 Processor Trace System

The debug system of the Cortex-M3 processor is based on the CoreSight architecture. The CoreSight-based designs enable the memory and peripheral registers to be examined even when the CPU is running. It also includes several trace capabilities:

- Data trace, generating events to record data reads/writes, exceptions/interrupts, and PC (program counter) sampling information.
- Software trace, supporting output of debug messages (for example, printf) to the host.
- Instruction trace, collecting a sequence of every executed instruction continuously for a selected portion of your application.

Trace data can be useful for debugging issues and collecting statistics:

- Locating errors that have irregular symptoms
- Analyzing dynamic system behavior
- Optimizing performance bottlenecks
- Counting code coverage statistics

Trace results are generated in the form of packets, which can be of various lengths. The trace components transfer the packets using the advanced trace bus (ATB) to the TPIU, which formats the packets into the trace interface protocol (TIP). The data is then captured by an external trace capture device such as a trace port analyzer (TPA).

The main components of the Cortex-M3 processor that can be a trace source:

- DWT, for data trace
- ITM, for software trace
- ETM, for full instruction trace

DWT, ITM, and ETM generate trace data in the form of packets and transfer them through the ATB to the TPIU.

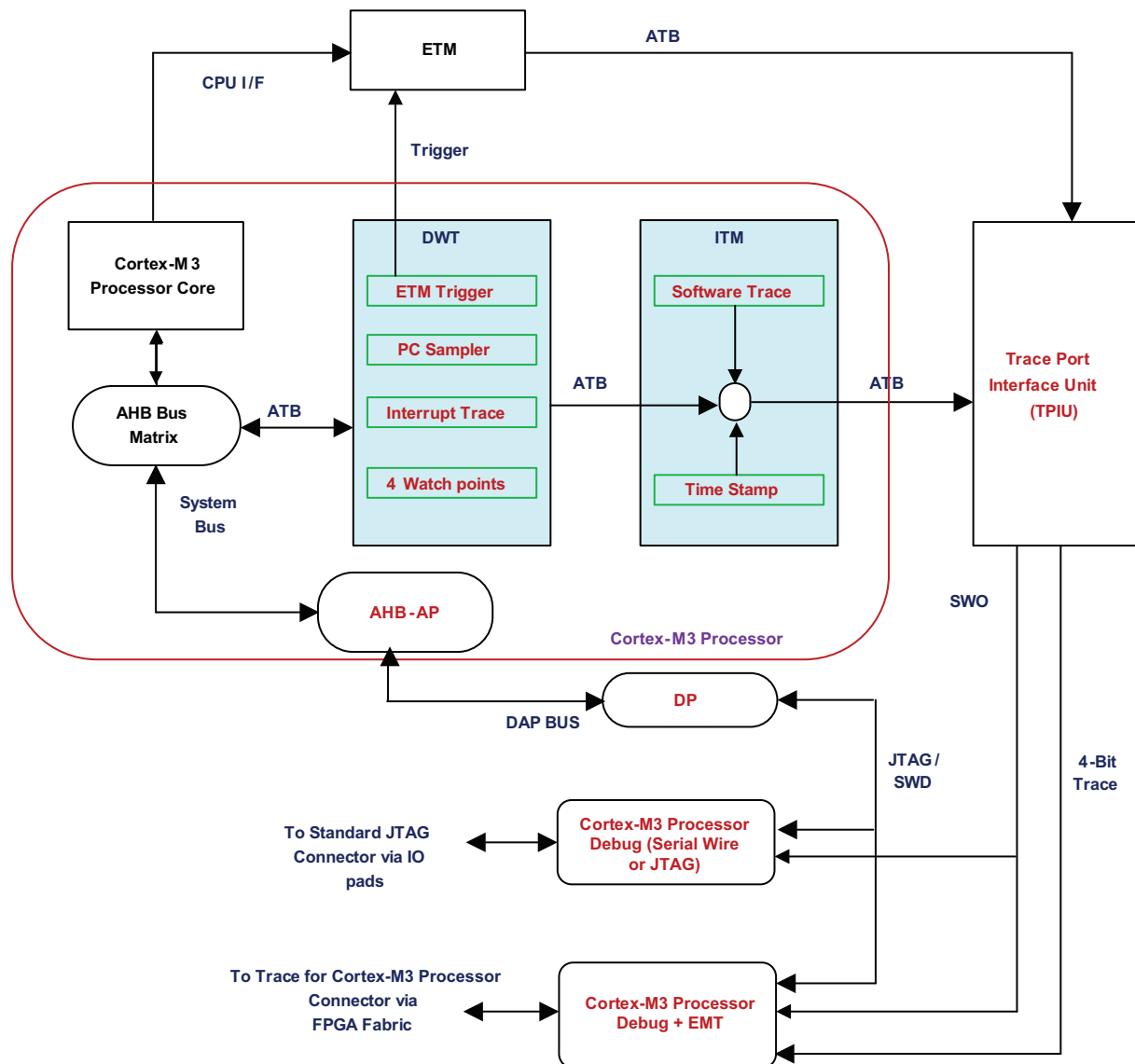
The TPIU has two operation modes:

- Clocked mode, using up to 4-bit (1-, 2- or 4-bit) parallel data outputs
- SWV mode, using the single-bit SWO format. Instruction trace from ETM must use the parallel trace port, while packets of data trace and software trace normally use SWO (called SWO trace) but can also be multiplexed with the ETM trace stream through the parallel trace port.

The following figure shows the diagram of a Cortex-M3 processor trace system. JTAG/SWD, SWO, and the 4-bit parallel trace port can be deployed into a 20-pin Cortex Debug + ETM connector on the target.

Note: The TDO signal of JTAG is multiplexed with SWO, so that SWO trace is not accessible when the DP is in a JTAG configuration. Only the SWD interface can be used together with SWO.

Figure 2 • Trace System Block Diagram



The following table shows pin multiplexing details for JTAG, SWD, and ETM modes of the debug section. For more details on pin information, refer to the [DS0115: SmartFusion2 Pin Descriptions Datasheet](#).

Table 3 • Signal Multiplexing

FPGA Pin	JTAG Mode	SWD Mode	ETM Mode
JTAG_TMS/ M3_TMS/ M3_SWDIO	TMS	SWDIO	SWDIO
JTAG_TCK/ M3_TCK	TCK	SWCLK	SWCLK

Table 3 • Signal Multiplexing (continued)

JTAG_TDO/ M3_TDO/ M3_SWO	TDO	SWO	SWO
JTAG_TDI/ M3_TDI	TDI		TRACECLK
			TRACEDATA[3:0]

2.5.2.1 Data Watch Point (DWP) and Trace

The DWT unit is able to provide either focused data trace or global data trace. It has four comparators used to compare the following conditions:

- Hardware watch point: generates a watch point event to the processor to invoke debug modes such as halt or debug monitor.
- ETM trigger: causes the ETM to emit a trigger packet in the instruction trace stream.
- PC sampler event trigger
- Data address sample trigger

2.5.2.2 Instrumentation Trace Macrocell

ITM provides the support for the debug message output, such as printf, and feeds output to the TPIU. ITM uses a FIFO to buffer the output messages and outputs are not delayed as UART transfers. The output messages can be collected at the TPI or the SWV interface on TPIU. ITM timestamps the outputs and it outputs the messages from the DWT unit.

2.5.2.3 Embedded Trace Macrocell

The ETM block is a high speed, low power-consumption debugging tool that provides instruction trace only, and which feeds output to the TPIU. The ETM has a FIFO queue of 24 bytes, and ETM outputs 8 bits of data at a time at the core clock speed. This output is compatible with the AMBA trace bus (ATB). The ETM trace is supported by tools like Keil Trace, IAR Trace, Greenhills software trace, and others. The ETM provides the following features:

- Tracing of 16-bit and 32-bit thumb instructions
- Four EmbeddedICE watchpoint inputs
- A Trace Start/Stop block with EmbeddedICE inputs
- Two external inputs
- Global time-stamping

2.6 Cortex-M3 Processor Port Descriptions

The following table lists all the ports related to the Cortex-M3 subsystem, their direction, and a description of the ports.

Table 4 • Port Details of the Cortex-M3-Subsystem

Port Name	Direction	Pad	Description
RXEV	In	No	Causes the Cortex-M3 to wake up from a wait for event (WFE) instruction. The event input, RXEV, is registered even when not waiting for an event, and so affects the next WFE.
TXEV	Out	No	Event transmitted as a result of a Cortex-M3 SEV (send event) instruction. This is a single-cycle pulse equal to 1 M3_CLK period.
SLEEP	Out	No	Signal is asserted when the Cortex-M3 processor is in sleep now or sleep-on-exit mode, and indicates that the clock to the processor can be stopped.

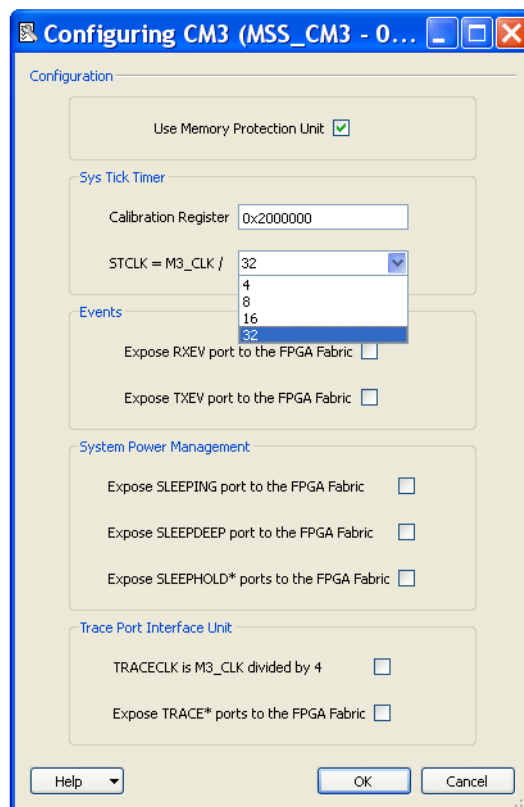
Table 4 • Port Details of the Cortex-M3-Subsystem (continued)

Port Name	Direction	Pad	Description
DEEPSLEEP	Out	No	Signal is asserted when the Cortex-M3 processor is in sleep now or sleep-on-exit mode when the SLEEPDEEP bit of the system control register is set.
SLEEPHOLDREQn	In	No	Request to extend Cortex-M3 processor sleep state. Signal is asserted when SLEEPING signal is High.
SLEEPHOLDACKn	Out	No	Signal is asserted to confirm the Cortex-M3 processor sleep state extension request.
TRACECLK	Out	No	TRACETRACEDATA changes on both the edges of TRACECLK.
TRACEDATA[3:0]	Out	No	Output data for clocked modes.

2.7 How to Use the Cortex-M3 Processor and the Debug Subsystem

2.7.1 Configuration Through Libero Software and Firmware

The Cortex-M3 processor and debug subsystem can be configured using the Libero® SoC design software. Using the MSS Cortex-M3 (CM3) configurator macro, various options can be selected, as shown in the following figure.

Figure 3 • CM3 Configurator

The timing arcs for interrupts to the Cortex-M3 sourced from the FPGA fabric have been updated in Libero SoC. In addition, timing arcs for the Cortex-M3 Embedded Trace Macrocell (ETM) have been added.

2.7.1.1 Memory Protection Unit

The MPU can be enabled by the selection option provided, as shown in the preceding figure. The following table lists all the registers that can be used to configure the MPU for the creation of the protected memory regions and setting the privileges for the created memory region in the firmware.

Table 5 • MPU Configuration Register

Name of Register	Access Type	Address	Reset Value
MPU type register	Read Only	0xE000ED90	0x800
MPU control register	Read/Write	0xE000ED94	0x0
MPU region number	Read/Write	0xE000ED98	NA
MPU region base address	Read/Write	0xE000ED9C	NA
MPU region attribute and size	Read/Write	0xE000EDA0	NA

2.7.1.2 SysTick Timer Configuration

The SysTick timer can be configured using the Libero software, as shown in the [Figure 3](#), page 16, for the SysTick calibration value; which is the rollover value of the internal SysTick timer, and SysTick clock frequency as the division (4, 8, 16, or 32) of Cortex-M3 clock. This value is loaded into the STCLK_DIVISOR register and it has to be configured to make sure that the SysTick clock frequency is less than half of the frequency of Cortex-M3. SysTick also can be configured using the firmware by using the following register, as depicted in the following table

Table 6 • SysTick Configuration Register

Name of Register	Access Type	Address	Reset value
SysTick Control & Status	Read/Write	0xE000E010	0x0
SysTick Reload value	Read/Write	0xE000E014	Unpredictable
SysTick Current Value	Read/Write clear	0xE000E018	Unpredictable
SysTick Calibration value	Read-only	0xE000E01C	STCALIB set through the Libero software

2.7.1.3 Events Configuration

TXEV and RXEV event signals of the Cortex-M3 processor can be exposed to the FPGA fabric. This can be configured using the Libero software, as shown in [Figure 3](#), page 16.

2.7.1.4 System Power Management Configuration

The Cortex-M3 processor provides various power modes. M3_CLK is gated off when in SLEEPING or SLEEPDEEP mode. SLEEPING and SLEEPDEEP signals are available at the FPGA fabric interface. Sleep mode extension handshake signals are available at the FPGA fabric interface. System power management options can be configured as shown in [Figure 3](#), page 16.

2.7.1.5 Trace Port Interface Unit (TPIU) Configuration

TRACECLK & TRACEDATA[3:0] can be exposed to the FPGA fabric. TACECLK can be configured for these signals by using the Libero software, as shown in the [Figure 3](#), page 16.

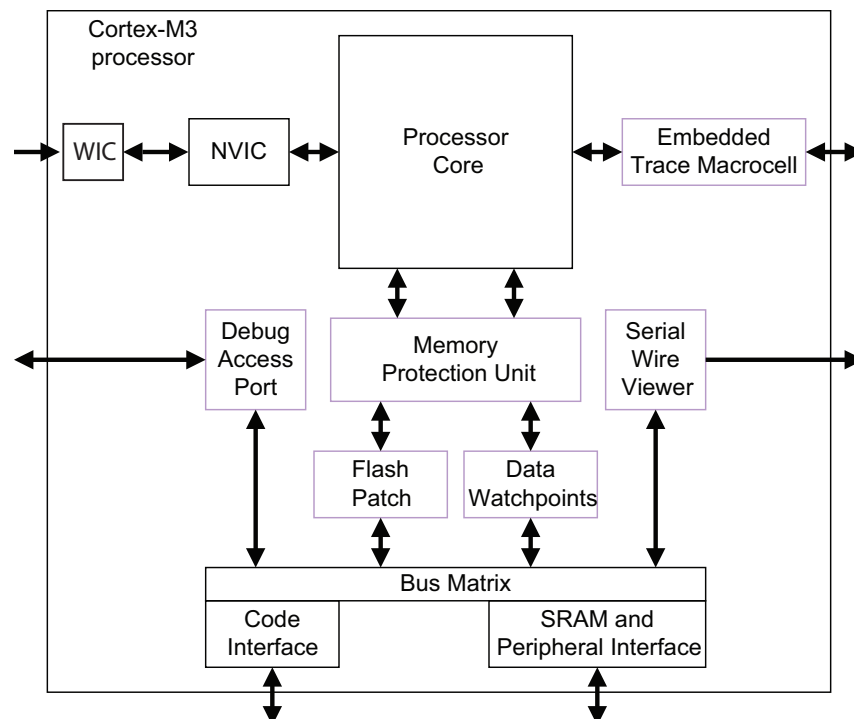
Note: If the user design is using the FPGA fabric based master, the Cortex-M3 processor requires a valid program in eNVM (from eNVM start address 0x60000000) to execute at power-up or power-on reset. The valid program can be a simple user boot code or a simple loop program. You can select a .hex file of a valid program for eNVM data client using the SystemBuilder.

3 Cortex-M3 Processor (Reference Material)

The Cortex-M3 processor is a high performance 32-bit processor designed for the microcontroller market. It offers significant benefits to developers, including:

- Outstanding processing performance combined with fast interrupt handling
- Enhanced system debug with extensive breakpoint and trace capabilities
- Efficient processor core, system, and memories
- Ultra-low power consumption with integrated Sleep modes
- Platform security robustness, with optional integrated memory protection unit (MPU)

Figure 4 • Cortex-M3 Processor Implementation



The Cortex-M3 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including a range of single-cycle and SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb® instruction set based on Thumb-2 technology, ensuring high code density and reduced program memory requirements. The Cortex-M3 processor instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M3 processor closely integrates a configurable nested interrupt controller (NVIC), to deliver industry-leading interrupt performance. The NVIC includes a non-maskable interrupt (NMI), and provides up to 256 interrupt priority levels. NVIC in SmartFusion2 SoC FPGA MSS is set to have 83 interrupts (including non-maskable interrupt). The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency.

This is achieved through the hardware stacking of registers, and the ability to suspend load-multiple and store-multiple operations. Interrupt handlers do not require wrapping in assembly code, removing any

code overhead from the ISRs. A Tail-chain optimization also significantly reduces the overhead when switching from one ISR to another.

To optimize low-power designs, the NVIC integrates with the sleep modes, that include a deep sleep function that enables the entire device to be rapidly powered down while still retaining program state.

3.1 System Level Interface

The Cortex-M3 processor provides multiple interfaces using AMBA® technology to provide high speed, low latency memory accesses. It supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks, and thread-safe Boolean data handling.

The Cortex-M3 processor has an optional memory protection unit (MPU) that provides fine grain memory control, enabling applications to utilize multiple privilege levels, separating and protecting code, data and stack on a task-by-task basis. Such requirements are becoming critical in many embedded applications such as automotive systems.

3.2 Integrated Configurable Debug

The Cortex-M3 processor implements a complete hardware debug solution. This provides high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin Serial Wire Debug (SWD) port that is ideal for microcontrollers and other small package devices. The MCU vendor determines the debug feature configuration and therefore this can differ across different devices and families.

For system trace the processor integrates an Instrumentation Trace Macrocell™ (ITM) alongside data watchpoints and a profiling unit. To enable simple and cost-effective profiling of the system events these generate, a Serial Wire Viewer (SWV) can export a stream of software-generated messages, data trace, and profiling information through a single pin.

The optional Embedded Trace Macrocell (ETM) delivers unrivaled instruction trace capture in an area far smaller than traditional trace units, enabling many low cost MCUs to implement full instruction trace for the first time.

The optional Flash Patch and Breakpoint Unit (FPB) provides up to eight hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to eight words in the program code in the CODE memory region. This enables applications stored on a non-erasable, ROM-based microcontroller to be patched if a small programmable memory, for example flash, is available in the device. During initialization, the application in ROM detects, from the programmable memory, whether a patch is required. If a patch is required, the application programs the FPB to remap a number of addresses. When those addresses are accessed, the accesses are redirected to a remap table specified in the FPB configuration, which means the program in the non-modifiable ROM can be patched.

3.3 Cortex-M3 Processor Features and Benefits Summary

- Tight integration of system peripherals reduces area and development costs.
- Thumb instruction set combines high code density with 32-bit performance.
- Code-patch ability for ROM system updates.
- Power control optimization of system components.
- Integrated Sleep modes for low power consumption.
- Fast code execution permits slower processor clock or increases sleep mode time.
- Hardware division and fast multiplier.
- Deterministic, high-performance interrupt handling for time-critical applications.
- Optional memory protection unit (MPU) for safety-critical applications.
- Extensive debug and trace capabilities—Serial Wire Debug and Serial Wire Trace reduce the number of pins required for debugging, tracing and code profiling.

3.4 Cortex-M3 Processor Core Peripherals

3.4.1 Nested Vectored Interrupt Controller

The Nested Vectored Interrupt Controller (NVIC) is an embedded interrupt controller that supports low latency interrupt processing.

3.4.2 System Control Block

The System control block (SCB) is the programmers model interface to the processor. It provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

3.4.3 System Timer

The system timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.

3.4.4 Memory Protection Unit

The Memory protection unit (MPU) improves system reliability by defining the memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region.

3.5 Cortex-M3 Processor Description

3.5.1 Programmers Model

This section describes the Cortex-M3 processor programmers model. In addition to the individual core register descriptions, it contains information about the processor modes and privilege levels for software execution and stacks.

3.5.1.1 Processor Mode and Privilege Levels for Software Execution

The processor modes are:

- **Thread mode:** Used to execute application software. The processor enters Thread mode when it comes out of reset.
- **Handler mode:** Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing.

The privilege levels for software execution are:

- **Unprivileged:** The software:
 - has limited access to the MSR and MRS instructions, and cannot use the CPS instruction
 - cannot access the system timer, NVIC, or system control block
 - might have restricted access to memory or peripherals.Unprivileged software executes at the unprivileged level.
- **Privileged:** The software can use all the instructions and has access to all resources. Privileged software executes at the privileged level.

In Thread mode, the CONTROL register controls whether software execution is privileged or unprivileged, see [CONTROL Register](#), page 27. In Handler mode, software execution is always privileged.

Only privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode. Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software.

3.5.1.2 Stacks

The processor uses a full descending stack. This means the stack pointer holds the address of the last stacked item in the stack memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements two stacks, the main stack and the process stack, held in independent registers, see [Stack Pointer](#), page 22.

In Thread mode, the CONTROL register controls whether the processor uses the main stack or the process stack, see [CONTROL Register](#), page 27. In Handler mode, the processor always uses the main stack. The options for processor operations are:

Table 7 • Summary of Processor Mode, Execution Privilege Level, and Stack Use Options

Processor mode	Used to execute	Privilege level for software execution	Stack used
Thread	Applications	Privileged or unprivileged	Main stack or process stack
Handler	Exception handlers	Always privileged	Main stack

3.5.1.3 Core Registers

The following figure shows the processor core registers.

Figure 5 • Core Register Set

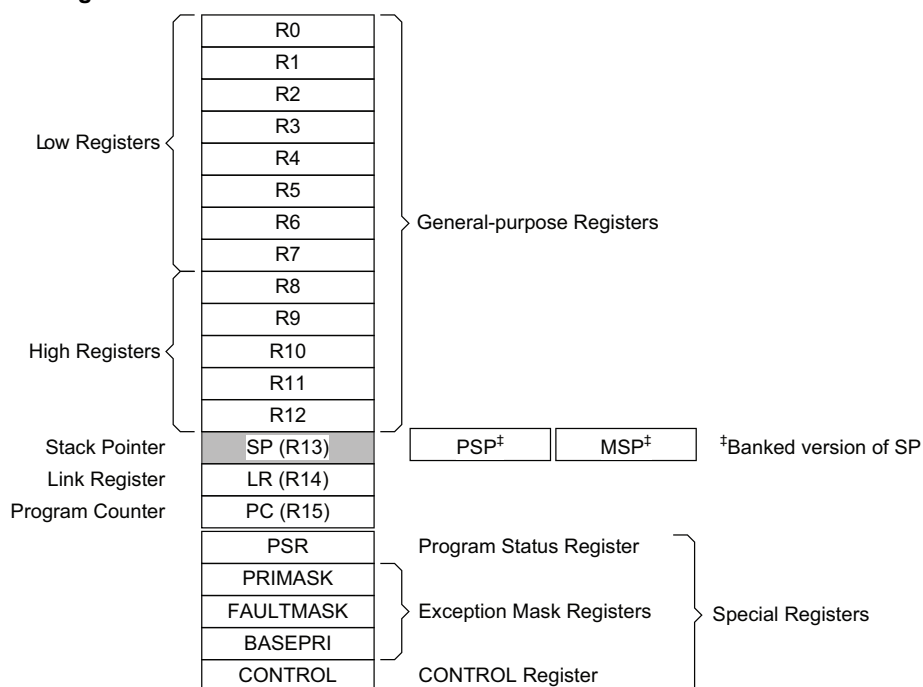


Table 8 • Core Register Set Summary

Name	Type ¹	Required privilege ²	Reset value	Description
R0-R12	RW	Either	Unknown	General-Purpose Registers
MSP	RW	Privileged	See description	Stack Pointer
PSP	RW	Either	Unknown	Stack Pointer
LR	RW	Either	0xFFFFFFFF	Link Register
PC	RW	Either	See description	Program Counter
PSR	RW	Privileged	Unknown	Program Status Register
ASPR	RW	Either	Unknown	Application Program Status Register
IPSR	RO	Privileged	0x00000000	Interrupt Program Status Register

Table 8 • Core Register Set Summary (continued)

Name	Type ¹	Required privilege ²	Reset value	Description
R0-R12	RW	Either	Unknown	General-Purpose Registers
EPSR	RO	Privileged	0x01000000	Execution Program Status Register
PRIMASK	RW	Privileged	0x00000000	Priority Mask Register
FAULTMASK	RW	Privileged	0x00000000	Fault Mask Register
BASEPRI	RW	Privileged	0x00000000	Base Priority Mask Register
CONTROL	RW	Privileged	0x00000000	CONTROL Register

1. Describes access type during program execution in Thread mode and Handler mode. Debug access can differ.

2. An entry of Either means privileged and unprivileged software can access the register.

The following sections describe these registers in detail.

3.5.1.3.1 General-Purpose Registers

R0-R12 are 32-bit general-purpose registers for data operations.

3.5.1.3.2 Stack Pointer

The Stack Pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use:

- 0: Main Stack Pointer (MSP). This is the reset value.
- 1: Process Stack Pointer (PSP)

On reset, the processor loads the MSP with the value from address 0x00000000.

3.5.1.3.3 Link Register

The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions. On reset, the processor sets the LR value to 0xFFFFFFFF.

3.5.1.3.4 Program Counter

The Program Counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit and must be 1.

3.5.1.3.5 Program Status Register

The Program Status Register (PSR) combines:

- Application Program Status Register (APSR)
- Interrupt Program Status Register (IPSR)
- Execution Program Status Register (EPSR)

These registers are mutually exclusive bitfields in the 32-bit PSR. The bit assignments are shown in the following figure.

Figure 6 • Program Status Register

	31	30	29	28	27	26	25	24	23	16	15	10	9	8	0
APSR	N	Z	C	V	Q	Reserved									
IPSR	Reserved										ISR_NUMBER				
EPSR	Reserved				ICI/IT	T	Reserved				ICI/IT	Reserved			

Access these registers individually or as a combination of any two or all three registers, using the register name as an argument to the MSR or MRS instructions. For example:

- Read all of the registers using PSR with the MRS instruction.
- Write to the APSR using APSR with the MSR instruction.

The following table shows the PSR combinations and attributes.

Table 9 • PSR Combinations and Attributes

Register	Type	Combination
PSR	RW ^{1,2}	APSR, EPSR, and IPSR
IEPSR	RO	EPSR and IPSR
IAPSR	RW ¹	APSR and IPSR
EAPSR	RW ²	APSR and EPSR

1. The processor ignores writes to the IPSR bits.
2. Reads of the EPSR bits return zero, and the processor ignores writes to the these bits.

See the instruction descriptions in [MRS](#), page 91 and [MSR](#), page 92 for more information about how to access the program status registers.

3.5.1.3.6 Application Program Status Register

The APSR contains the current state of the condition flags from previous instruction executions. See the register summary in the following table for its attributes. The following table lists the bit assignments.

Table 10 • Application Program Status Register

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	Saturation flag
[27:0]	-	Reserved

3.5.1.3.7 Interrupt Program Status Register

The IPSR contains the exception type number of the current Interrupt Service Routine (ISR). See the register summary in [Table 8](#), page 21 for its attributes. The following table lists the bit assignments.

Table 11 • IPSR Bit Assignments

Bits	Name	Function
[31:9]		Reserved

Table 11 • IPSR Bit Assignments

Bits	Name	Function
[8:0]	ISR_NUMBER	This is the number of the current exception: 0 = Thread mode 1 = Reserved 2 = NMI 3 = HardFault 4 = MemManage 5 = BusFault 6 = UsageFault 7-10 = Reserved 11 = SVCall 12 = Reserved for Debug 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0. . . . 255 = IRQ239 See Exception Types , page 37 for more information

3.5.1.3.8 Execution Program Status Register

The EPSR contains the Thumb state bit, and the execution state bits for either the:

- If-Then (IT) instruction
- Interruptible-Continuable Instruction (ICI) field for an interrupted load multiple or store multiple instruction.

See the register summary in [Table 8](#), page 21 for the EPSR attributes. The following table lists the bit assignments.

Table 12 • EPSR Bit Assignments

Bits	Name	Function
[31:27]		Reserved.
[26:25], [15:10]	ICI/IT	Indicates the interrupted position of a continuable instruction, or the execution state of an IT instruction (see IT , page 85).
[24]	T	Thumb state bit.
[23:16]		Reserved.
[9:0]		Reserved.

Attempts to read the EPSR directly using the MRS instruction always return zero. Attempts to write the EPSR using the MSR instruction are ignored.

Interruptible-continuable Instructions

When an interrupt occurs during the execution of an LDM, STM, PUSH, or POP instruction, the processor:

- Stops the load multiple or store multiple instruction operation temporarily
- Stores the next register operand in the multiple operation to EPSR bits[15:12]

After servicing the interrupt, the processor:

- Continues loading the register pointed to by bits[15:12]
- Resumes execution of the multiple load or store instruction

When the EPSR holds ICI execution state, bits[26:25,11:10] are zero.

If-Then Block

The If-Then block contains up to four instructions following an IT instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See [IT](#), page 85 for more information.

Thumb State

The Cortex-M3 processor only supports execution of instructions in Thumb state. The following can clear the T bit to 0:

- instructions BLX, BX and POP{PC}
- restoration from the stacked xPSR value on an exception return
- bit[0] of the vector value on an exception entry or reset.

Attempting to execute instructions when the T bit is 0 results in a fault or lockup. See [Lockup](#), page 45 for more information.

The T bit can be modified both by software, using the mechanisms described in this section, and directly by the debugger.

3.5.1.3.9 Exception Mask Registers

The exception mask registers disable the handling of exceptions by the processor. Disable exceptions where they might impact on timing critical tasks.

To access the exception mask registers use the MSR and MRS instructions, or the CPS instruction to change the value of PRIMASK or FAULTMASK. See [MRS](#), page 91, [MSR](#), page 92, and [CPS](#), page 89 for more information.

Priority Mask Register

The PRIMASK register prevents activation of all exceptions with configurable priority. See the register summary in [Table 8](#), page 21 for its attributes. The following figure for bit assignments for MSR or MRS access.

Figure 7 • Priority Mask Register

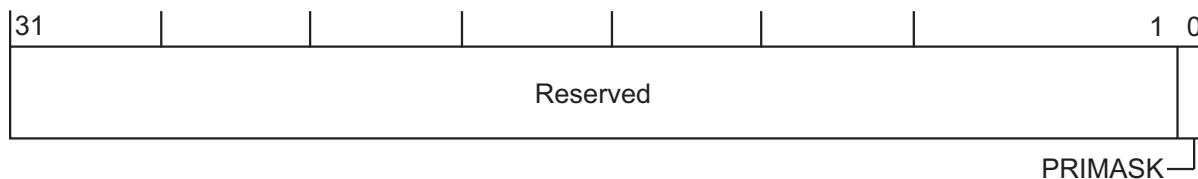


Table 13 • PRIMASK Register Bit Assignments

Bits	Name	Function
[31:1]		Reserved
[0]	PRIMASK	0: no effect 1: prevents the activation of all exceptions with configurable priority.

Fault Mask Register

The FAULTMASK register prevents activation of all exceptions except for Non-Maskable Interrupt (NMI). See the register summary in [Table 8](#), page 21 for its attributes.

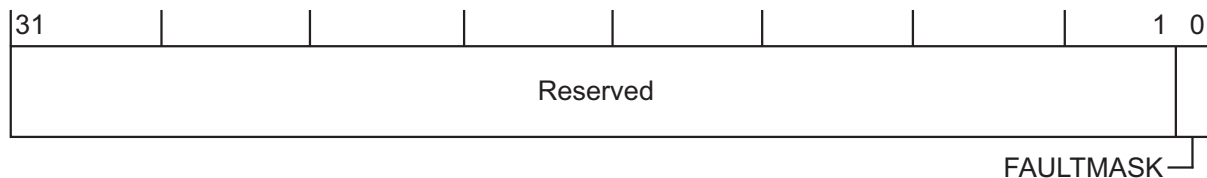


Figure 8 • Fault Mask Register

The following table lists the big assignments for MSR or MRS access.

Table 14 • FAULT Register Bit Assignments

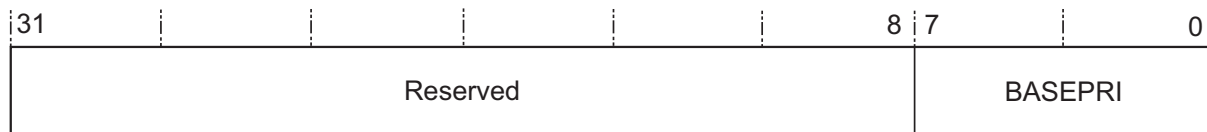
Bits	Name	Function
[31:1]	-	Reserved
[0]	FAULTMASK	0: no effect 1: prevents the activation of all exceptions except for NMI.

The processor clears the FAULTMASK bit to 0 on exit from any exception handler except the NMI handler.

Base Priority Mask Register

The BASEPRI register defines the minimum priority for exception processing. When BASEPRI is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the BASEPRI value. See the register summary in [Table 8](#), page 21 for its attributes.

Figure 9 • Base Priority Mask Register



The following table lists the big assignments for MSR or MRS access.

Table 15 • BASEPRI Register Bit Assignments

Bits	Name	Function
[31:8]		Reserved
[7:0]	BASEPRI ¹	Priority mask bits: 0x00: no effect Nonzero: defines the base priority for exception processing. The processor does not process any exception with a priority value greater than or equal to BASEPRI.

1. This field is similar to the priority fields in the interrupt priority registers. The device implements only bits[7:M] of this field, bits [M-1:0] read as zero and ignore writes. See [Interrupt Priority Registers](#), page 99 for more information. Remember that higher priority field values correspond to lower exception priorities.

3.5.1.3.10 CONTROL Register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode. Refer to the register summary in [Table 8](#), page 21 for its attributes. The following figure shows the bit assignments for MSR or MRS access.

Figure 10 • Control Register

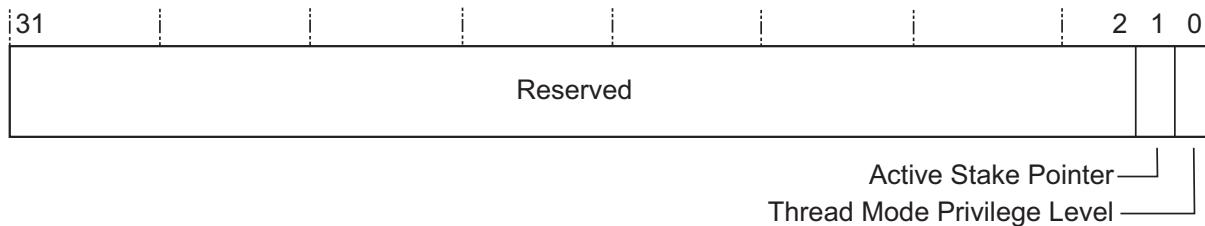


Table 16 • Control Register Bit Assignments

Bits	Name	Function
[31:2]		Reserved
[1]	Active stack pointer	Defines the currently active stack pointer: 0: MSP is the current stack pointer 1: PSP is the current stack pointer. In Handler mode this bit reads as zero and ignores writes.
[0]	Thread mode privilege level	Defines the Thread mode privilege level: 0: Privileged 1: Unprivileged.

Handler mode always uses the MSP, so the processor ignores explicit writes to the active stack pointer bit of the CONTROL register when in Handler mode. The exception entry and return mechanisms automatically update the CONTROL register based on the EXC_RETURN value, see [Table 23](#), page 43.

In an OS environment, ARM recommends that threads running in Thread mode use the process stack and the kernel and exception handlers use the main stack.

By default, Thread mode uses the MSP. To switch the stack pointer used in Thread mode to the PSP, either:

- use the MSR instruction to set the Active stack pointer bit to 1, see [MSR](#), page 92
- perform an exception return to Thread mode with the appropriate EXC_RETURN value, see [Table 23](#), page 43.

Note: When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB instruction execute using the new stack pointer. Refer to [ISB](#), page 91.

3.5.1.4 Exceptions and Interrupts

The Cortex-M3 processor supports interrupts and system exceptions. The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses Handler mode to handle all exceptions except for reset. Refer to [Exception Entry](#), page 41 and [Exception Return](#), page 42 for more information.

The NVIC registers control interrupt handling. See [Nested Vectored Interrupt Controller](#), page 20 for more information.

The following sections provide more information about the CMSIS

- [Power Management Programming Hints](#), page 47
- [CMSIS Functions](#), page 50
- [Accessing the Cortex-M3 Processor NVIC Registers Using CMSIS](#), page 96
- [NVIC programming hints](#), page 101

3.5.1.5 Data types

The processor:

- supports the following data types:
 - 32-bit words
 - 16-bit halfwords
 - 8-bit bytes.
- manages all data memory accesses as little-endian or big-endian. Instruction memory and Private Peripheral Bus (PPB) accesses are always performed as little-endian. The Cortex-M3 processor configured for SmartFusion2 SoC FPGA MSS uses only little-endian. Refer to [Memory Regions, Types and Attributes](#), page 29.

3.5.1.6 The Cortex Microcontroller Software Interface Standard

For a Cortex-M3 processor system, the Cortex Microcontroller Software Interface Standard (CMSIS) defines:

- a common way to:
 - access peripheral registers
 - define exception vectors
- the names of:
 - the registers of the core peripherals
 - the core exception vectors
- a device-independent interface for RTOS kernels, including a debug channel.

The CMSIS includes address definitions and data structures for the core peripherals in the Cortex-M3 processor. It also includes optional interfaces for middleware components comprising a TCP/IP stack and a Flash file system.

CMSIS simplifies software development by enabling the reuse of template code and the combination of CMSIS-compliant software components from various middleware vendors. Software vendors can expand the CMSIS to include their peripheral definitions and access functions for those peripherals.

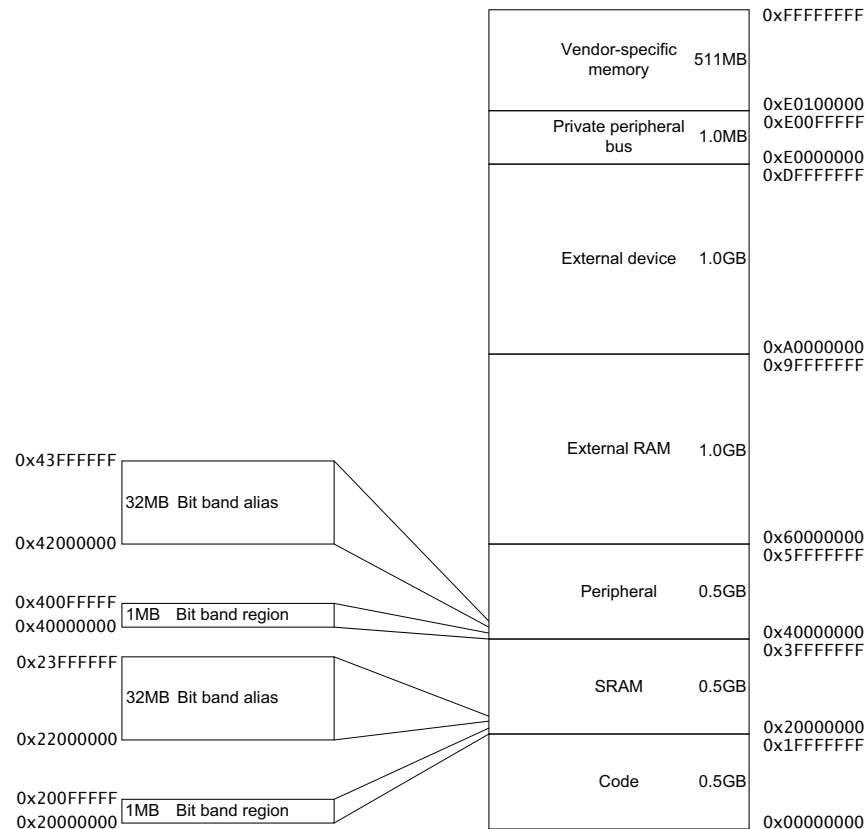
This document includes the register names defined by the CMSIS, and gives short descriptions of the CMSIS functions that address the processor core and the core peripherals.

Note: This document uses the register short names defined by the CMSIS. In a few cases these differ from the architectural short names that might be used in other documents.

3.5.2 Memory Model

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4GB of addressable memory. The following illustration shows the processor memory map.

Figure 11 • Processor Memory Map



The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data, see [Bit-Banding](#), page 32.

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers, see [Cortex-M3 Processor Peripherals](#), page 95.

3.5.2.1 Memory Regions, Types and Attributes

The memory map and the programming of the MPU splits the memory map into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

Normal: The processor can re-order transactions for efficiency, or perform speculative reads.

Device: The processor preserves transaction order relative to other transactions to Device or Strongly-ordered memory.

Strongly-ordered: The processor preserves transaction order relative to all other transactions Strongly-Ordered or Device.

The different ordering requirements for Device and Strongly-ordered memory mean that the memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

The additional memory attributes include:

Shareable: For a shareable memory region, the memory system provides data synchronization between bus masters in a system with multiple bus masters, for example, a processor with a DMA controller.

Strongly-ordered memory is always shareable.

If multiple bus masters can access a non-shareable memory region, software must ensure data coherency between the bus masters.

Execute Never (XN): Means the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction executed from an XN region.

3.5.2.2 Memory System Ordering of Memory Accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing this does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions.

However, the memory system does guarantee some ordering of accesses to Device and Strongly-ordered memory. The following figure shows the ordering of the memory accesses caused by two instructions A1 and A2 if A1 occurs before A2 in program order.

Figure 12 • Memory Ordering Restrictions

A1 \ A2		Normal access	Device access		Strongly-ordered access
			Non-shareable	Shareable	
Normal access		-	-	-	-
Device access, non-shareable		-	<	-	<
Device access, shareable		-	-	<	<
Strongly-ordered access		-	<	<	<

Where:

- Means that the memory system does not guarantee the ordering of the accesses.
- < Means that accesses are observed in program order, that is, A1 is always observed before A2.

3.5.2.3 Behavior of Memory Accesses

The following table provides information about the behavior of accesses to each region in the memory map.

Table 17 • Memory Access Behavior

Address range	Memory region	Memory Type ¹	XN ¹	Description
0x00000000-0x1FFFFFFF	Code	Normal		Executable region for program code. You can also put data here.
0x20000000-0x3FFFFFFF	SRAM	Normal		Executable region for data. You can also put code here. This region includes bit band and bit band alias areas, see Table 19 , page 32.
0x40000000-0x5FFFFFFF	Peripheral	Device	XN	This region includes bit band and bit band alias areas, see Table 20 , page 32.
0x60000000-0x9FFFFFFF	External RAM	Normal		Executable region for data.
0xA0000000-0xDFFFFFFF	External device	Device	XN	External Device memory.

Table 17 • Memory Access Behavior (continued)

Address range	Memory region	Memory Type ¹	XN ¹	Description
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly-ordered	XN	This region includes the NVIC, System timer, and system control block.
0xE0100000-0xFFFFFFFF	Vendor specific	Device	XN	Accesses to this region are to vendor-specific peripherals.

1. See [Memory Regions, Types and Attributes](#), page 29 for more information.

The Code, SRAM, and external RAM regions can hold programs. However, ARM recommends that programs always use the Code region. This is because the processor has separate buses that enable instruction fetches and data accesses to occur simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see [Memory Protection Unit](#), page 20.

3.5.2.3.1 Additional memory access constraints for caches and shared memory

When a system includes caches or shared memory, some memory regions have additional access constraints, and some regions are subdivided, as detailed in the following table.

Table 18 • Memory Region Shareability and Cache Policies

Address range	Memory region	Memory type ¹	Shareability	Cache policy ²
0x00000000- 0x1FFFFFFF	Code	Normal		WT
0x20000000- 0x3FFFFFFF	SRAM	Normal		WBWA
0x40000000- 0x5FFFFFFF	Peripheral	Device		
0x60000000- 0x7FFFFFFF	External RAM	Normal		WBWA
0x80000000- 0x9FFFFFFF				WT
0xA0000000- 0xBFFFFFFF	External device	Device	Shareable	
0xC0000000- 0xDFFFFFFF			Non-shareable	
0xE0000000- 0xE00FFFFF	Private Peripheral Bus	Strongly-ordered	Shareable	
0xE0100000- 0xFFFFFFFF	Vendor-specific device	Device		

1. See [Memory Regions, Types and Attributes](#), page 29 for more information.
2. WT = Write through, no write allocate. WBWA = Write back, write allocate.

3.5.2.3.2 Instruction Prefetch and Branch Prediction

The Cortex-M3 processor:

- Prefetches instructions ahead of execution
- Speculatively prefetches from branch target addresses.

3.5.2.4 Software Ordering of Memory Accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:

- the processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- the processor has multiple bus interfaces
- memory or devices in the memory map have different wait states
- some memory accesses are buffered or speculative.

[Memory System Ordering of Memory Accesses](#), page 30 describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

DMB: The Data Memory Barrier (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions. See [DMB](#), page 90.

DSB: The Data Synchronization Barrier (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute. See [DSB](#), page 91.

ISB: The Instruction Synchronization Barrier (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions. See [ISB](#), page 91.

MPU programming

Use a DSB followed by an ISB instruction or exception return to ensure that the new MPU configuration is used by subsequent instructions.

3.5.2.5 Bit-Banding

A bit-band region maps each word in a bit-band alias region to a single bit in the bit-band region. The bit-band regions occupy the lowest 1MB of the SRAM and peripheral memory regions.

Note: The Cortex-M3 processor does not support exclusive accesses to bit-band regions.

The memory map has two 32MB alias regions that map to two 1MB bit-band regions:

- accesses to the 32MB SRAM alias region map to the 1MB SRAM bit-band region, as detailed in [Table 19](#), page 32.
- accesses to the 32MB peripheral alias region map to the 1MB peripheral bit-band region, as detailed in [Table 20](#), page 32.

Table 19 • SRAM Memory Bit-banding Regions

Address Range	Memory SRAM Region	Instruction and Data Accesses
0x20000000-0x200FFFFF	Bit-band region	Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit addressable through bit-band alias.
0x22000000-0x23FFFFFF	Bit-band alias	Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped.

Table 20 • Peripheral Memory Bit-banding Regions

Address range	Memory SRAM Region	Instruction and Data Accesses
0x40000000-0x400FFFFF	Bit-band alias	Direct accesses to this memory range behave as peripheral memory accesses, but this region is also bit addressable through bit-band alias.
0x42000000-0x43FFFFFF	Bit-band region	Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not permitted.

Notes:

- A word access to the SRAM or peripheral bit-band alias regions maps to a single bit in the SRAM or peripheral bit-band region.
- Bit band accesses can use byte, halfword, or word transfers. The bit band transfer size matches the transfer size of the instruction making the bit band access.

The following formula shows how the alias region maps onto the bit-band region:

- $\text{bit_word_offset} = (\text{byte_offset} \times 32) + (\text{bit_number} \times 4)$
- $\text{bit_word_addr} = \text{bit_band_base} + \text{bit_word_offset}$

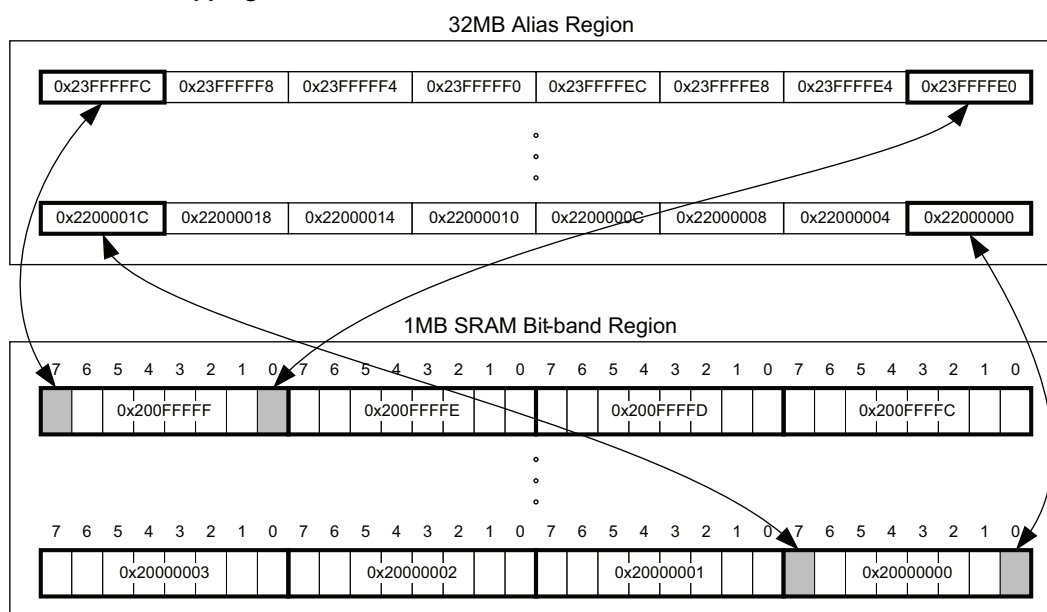
where:

- Bit_word_offset is the position of the target bit in the bit-band memory region.
- Bit_word_addr is the address of the word in the alias memory region that maps to the targeted bit.
- Bit_band_base is the starting address of the alias region.
- Byte_offset is the number of the byte in the bit-band region that contains the targeted bit.
- Bit_number is the bit position, 0-7, of the targeted bit.

The following illustration shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region.

- The alias word at 0x23FFFFE0 maps to bit[0] of the bit-band byte at 0x200FFFFF:
 $0x23FFFFE0 = 0x22000000 + (0xFFFF \times 32) + (0 \times 4)$.
- The alias word at 0x23FFFFFC maps to bit[7] of the bit-band byte at 0x200FFFFF:
 $0x23FFFFFC = 0x22000000 + (0xFFFF \times 32) + (7 \times 4)$.
- The alias word at 0x22000000 maps to bit[0] of the bit-band byte at 0x20000000:
 $0x22000000 = 0x22000000 + (0 \times 32) + (0 \times 4)$.
- The alias word at 0x2200001C maps to bit[7] of the bit-band byte at 0x20000000:
 $0x2200001C = 0x22000000 + (0 \times 32) + (7 \times 4)$.

Figure 13 • Bit-band Mapping



3.5.2.5.1 Directly Accessing an Alias Region

Writing to a word in the alias region updates a single bit in the bit-band region.

Bit[0] of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit[0] set to 1 writes a 1 to the bit-band bit, and writing a value with bit[0] set to 0 writes a 0 to the bit-band bit.

Bits[31:1] of the alias word have no effect on the bit-band bit. Writing 0x01 has the same effect as writing 0xFF. Writing 0x00 has the same effect as writing 0x0E.

Reading a word in the alias region:

- 0x00000000 indicates that the targeted bit in the bit-band region is set to zero
- 0x00000001 indicates that the targeted bit in the bit-band region is set to 1.

3.5.2.5.2 Directly accessing a bit-band region

Behavior of Memory Accesses, page 30 describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

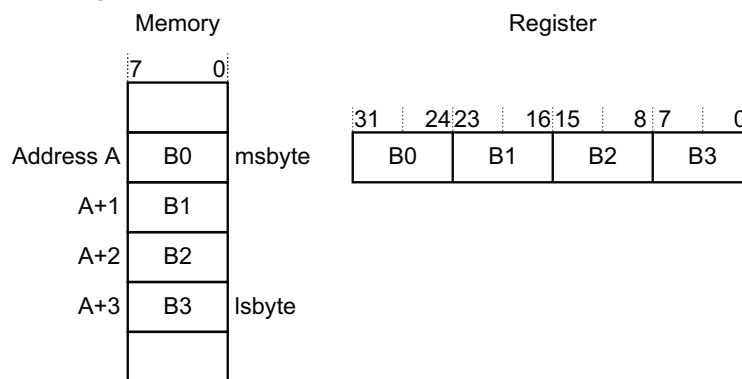
3.5.2.6 Memory Endianness

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. Byte-invariant big-endian format or Little-endian format describes how words of data are stored in memory.

3.5.2.6.1 Byte-invariant Big-endian Format

In byte-invariant big-endian format, the processor stores the most significant byte of a word at the lowest-numbered byte, and the least significant byte at the highest-numbered byte. The following illustration shows the byte-invariant big-endian format.

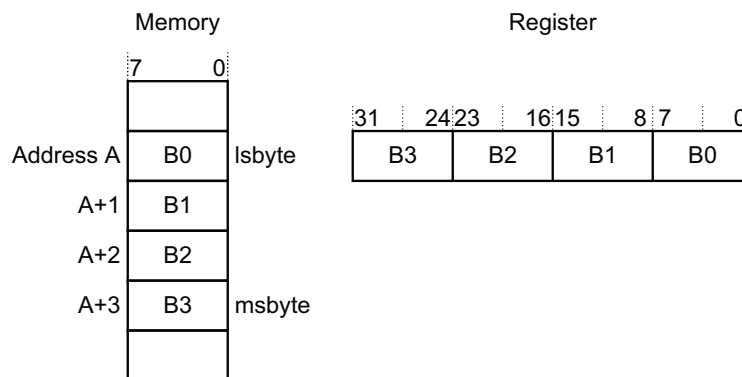
Figure 14 • Byte-Invariant Big-Endian Format



3.5.2.6.2 Little-Endian format

In little-endian format, the processor stores the least significant byte of a word at the lowest-numbered byte, and the most significant byte at the highest-numbered byte. Cortex-M3 processor configured for SmartFusion2 SoC FPGA MSS uses only little endian. The following figure illustrates the little-endian format.

Figure 15 • Little Endian Format



3.5.2.7 Synchronization Primitives

The Cortex-M3 processor instruction set includes pairs of synchronization primitives. These provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use them to perform a guaranteed read-modify-write memory update sequence, or for a semaphore mechanism.

A pair of synchronization primitives comprises:

3.5.2.7.1 A Load-Exclusive Instruction

Used to read the value of a memory location, requesting exclusive access to that location.

3.5.2.7.2 A Store-Exclusive Instruction

Used to attempt to write to the same memory location, returning a status bit to a register. If this bit is:

0: it indicates that the thread or process gained exclusive access to the memory, and the write succeeds.

1: it indicates that the thread or process did not gain exclusive access to the memory, and no write is performed.

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- the word instructions LDREX and STREX
- the halfword instructions LDREXH and STREXH
- the byte instructions LDREXB and STREXB.

Software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform a guaranteed read-modify-write of a memory location, software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Update the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location, and tests the returned status bit. If this bit is:
 - 0: The read-modify-write completed successfully.
 - 1: No write was performed. This indicates that the value returned at step 1 might be out of date. The software must retry the read-modify-write sequence.

Software can use the synchronization primitives to implement a semaphores as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.
2. If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.
3. If the returned status bit from step2 indicates that the Store-Exclusive succeeded then the software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed step 1.

The Cortex-M3 processor includes an exclusive access monitor, that tags the fact that the processor has executed a Load-Exclusive instruction. If the processor is part of a multiprocessor system, the system also globally tags the memory locations addressed by exclusive accesses by each processor.

The processor removes its exclusive access tag if:

- It executes a CLREX instruction.
- It executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs. This means the processor can resolve semaphore conflicts between different threads.

In a multiprocessor implementation:

- executing a CLREX instruction removes only the local exclusive access tag for the processor
- executing a Store-Exclusive instruction, or an exception. removes the local exclusive access tags, and all global exclusive access tags for the processor.

For more information about the synchronization primitive instructions, see [LDREX and STREX](#), page 66 and [CLREX](#), page 67.

3.5.2.8 Programming Hints for the Synchronization Primitives

ISO/IEC C cannot directly generate the exclusive access instructions. Some CMSIS provides intrinsic functions for generation of these instructions. The following table lists the functions that CMSIS provides.

Table 21 • CMSIS Functions for Exclusive Access Instructions

Instruction	Intrinsic Function
LDREX, LDREXH, or LDREXB	unsigned char __LDREXB(volatile char *ptr) unsigned short __LDREXH(volatile short *ptr) unsigned int __LDREXB(volatile int *ptr)
STREX, STREXH, or STREXB	int __STREXB(unsigned char val, volatile char *ptr) int __STREXB(unsigned short val, volatile short *ptr) int __STREXB(unsigned int val, volatile int *ptr)
CLREX	void __CLREX(void)

The actual exclusive access instruction generated depends on the data type of the pointer passed to the intrinsic function.

3.5.3 Exception Model

This section describes the exception model.

3.5.3.1 Exception States

Each exception is in one of the following states:

Inactive: The exception is not active and not pending.

Pending: The exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.

Active: An exception that is being serviced by the processor but has not completed. An exception handler can interrupt the execution of another exception handler. In this case both exceptions are in the active state.

Active and pending: The exception is being serviced by the processor and there is a pending exception from the same source.

3.5.3.2 Exception Types

The exception types are:

Reset: Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.

NMI: A Non-Maskable Interrupt (NMI) can be signaled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled and has a fixed priority of -2. NMIs cannot be:

- masked or prevented from activation by any other exception.
- preempted by any exception other than Reset.

HardFault: A HardFault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. HardFaults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.

MemManage: A MemManage fault is an exception that occurs because of a memory protection related fault. The MPU or the fixed memory protection constraints determines this fault, for both instruction and data memory transactions. This fault is always used to abort instruction accesses to Execute Never (XN) memory regions.

BusFault: A BusFault is an exception that occurs because of a memory related fault for an instruction or data memory transaction. This might be from an error detected on a bus in the memory system.

UsageFault: A UsageFault is an exception that occurs because of a fault related to instruction execution.

This includes:

- an undefined instruction
- an illegal unaligned access
- invalid state on instruction execution
- an error on exception return.

The following can cause a UsageFault when the core is configured to report them:

- an unaligned address on word and halfword memory access
- division by zero.

SVC: A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.

PendSV: PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.

SysTick: A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

Interrupt (IRQ): A interrupt, or IRQ, is an exception signaled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Table 22 • Properties of the Different Exception Types

Exception number ¹	IRQ number ¹	Exception type	Priority	Vector address or offset ²	Activation
1		Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	HardFault	-1	0x0000000C	
4	-12	MemManage	Configurable	0x00000010	Synchronous
5	-11	BusFault	Configurable ³	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	UsageFault	Configurable ³	0x00000018	Synchronous
7-10		Reserved			
11	-5	SVCall	Configurable ³	0x0000002C	Synchronous
12-13		Reserved			
14	-2	PendSV	Configurable ³	0x00000038	Asynchronous
15	-1	SysTick	Configurable ³	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable ⁴	0x00000040 and above ⁵	Asynchronous

1. To simplify the software layer, the CMSIS only uses IRQ numbers and therefore uses negative values for exceptions other than interrupts. The IPSR returns the Exception number, see [Interrupt Program Status Register](#), page 23.
2. See [Vector Table](#), page 40 for more information.
3. See [System Handler Priority Registers](#), page 110.
4. See [Interrupt Priority Registers](#), page 99.
5. Increasing in steps of 4.

Privileged software can disable the exceptions that [Table 22](#), page 39 shows as having configurable priority. See [System Handler Control and State Register](#), page 112 and [Interrupt Clear-enable Registers](#), page 97.

For more information about HardFaults, MemManage faults, BusFaults, and UsageFaults, see [Fault Handling](#), page 43.

3.5.3.3 Exception Handlers

The processor handles exceptions using:

Interrupt Service Routines (ISRs): Interrupts IRQ0 to IRQ239 are the exceptions handled by ISRs.

Fault handlers: HardFault, MemManage, UsageFault and BusFault are fault exceptions handled by the fault handlers.

System handlers: NMI, PendSV, SVCall SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

3.5.3.4 Vector Table

The vector table contains the reset value of the stack pointer, and the start addresses, also called exception vectors, for all exception handlers. Figure 5, page 21 shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code.

Exception number	IRQ number	Offset	Vector
255	239	0x03FC	IRQ239
.	.	.	.
.	.	.	.
18	2	0x004C	IRQ2
17	1	0x0048	IRQ1
16	0	0x0044	IRQ0
15	-1	0x0040	Systick
14	-2	0x003C	PendSV
13		0x0038	Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

Figure 16 • Vector Table

On system reset, the vector table is fixed at address 0x00000000. Privileged software can write to the VTOR to relocate the vector table start address to a different memory location, in the range 0x00000080 to 0x3FFFFFF80, see [Vector Table Offset Register](#), page 106.

3.5.3.5 Exception Priorities

As [Table 22](#), page 39 shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, HardFault, and NMI.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see [System Handler Priority Registers](#), page 110 and [Interrupt Priority Registers](#), page 99.

Note: Configurable priority values are in the range 0-255. This means that the Reset, HardFault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

3.5.3.6 Interrupt Priority Grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This divides each interrupt priority register entry into two fields:

- An upper field that defines the *group priority*
- A lower field that defines a *subpriority* within the group.

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler,

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see [Application Interrupt and Reset Control Register](#), page 106.

3.5.3.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

Preemption: When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See [Interrupt Priority Grouping](#), page 41 for more information about preemption by an interrupt.

When one exception preempts another, the exceptions are called nested exceptions. See [Exception Entry](#), page 41 more information.

Return: This occurs when the exception handler is completed, and:

- there is no pending exception with sufficient priority to be serviced
- the completed exception handler was not handling a late-arriving exception.

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See [Exception Return](#), page 42 for more information.

Tail-chaining: This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.

Late-arriving: This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

3.5.3.7.1 Exception Entry

Exception entry occurs when there is a pending exception with sufficient priority and either:

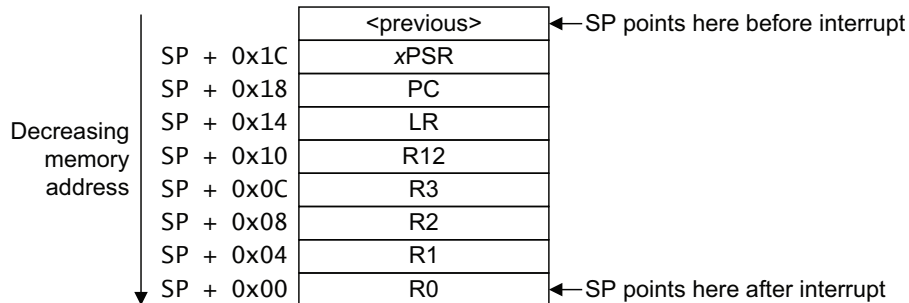
- The processor is in Thread mode
- The new exception is of higher priority than the exception being handled, in which case the new exception preempts the exception being handled.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has greater priority than any limit set by the mask register, see [Exception Mask Registers](#), page 25. An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as *stacking* and the structure of eight data words is referred to as a *stack frame*. The following figure illustrates the information contained in the stack frame.

Figure 17 • Exception Entry Stack Contents



Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. The alignment of the stack frame is controlled via the STKALIGN bit of the *Configuration Control Register* (CCR).

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

The processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the LR. This indicates which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

3.5.3.7.2 Exception Return

Exception return occurs when the processor is in Handler mode and execution of one of the following instructions attempts to set the PC to an EXC_RETURN value:

- an LDM or POP instruction that loads the PC
- an LDR instruction with PC as the destination
- a BX instruction using any register.

The processor saves an EXC_RETURN value to the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. Bits[31:4] of an EXC_RETURN value are 0xFFFFFFFF.

When the processor loads a value matching this pattern to the PC it detects that the operation is a not a normal branch operation and, instead, that the exception is complete. Therefore, it starts the exception return sequence. Bits[3:0] of the EXC_RETURN value indicate the required return stack and processor mode, as noted in the following table.

Table 23 • Exception Return Behavior

EXC_RETURN	Description
0xFFFFFFFF1	Return to Handler mode. Exception return gets state from the main stack. Execution uses MSP after return.
0xFFFFFFFF9	Return to Thread mode. Exception Return get state from the main stack. Execution uses MSP after return.
0xFFFFFFFFD	Return to Thread mode. Exception return gets state from the process stack. Execution uses PSP after return.
All other values	Reserved.

3.5.4 Fault Handling

Faults are a subset of the exceptions, see [Exception Model](#), page 37. The following generates a fault:

- a bus error on:
 - an instruction fetch or vector table load
 - a data access
- an internally-detected error such as an undefined instruction
- attempting to execute an instruction from a memory region marked as *Non-Executable* (XN).
- attempting to execute an instruction while the EPSR T-bit is clear. For example, as the result of an erroneous BX instruction, or a vector fetch from a vector table entry with bit[0] clear.
- an MPU fault because of a privilege violation or an attempt to access an unmanaged region.

3.5.4.1 Fault Types

The following table shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates that the fault has occurred. See [Configurable Fault Status Register](#), page 113 for more information about the fault status registers.

Table 24 • Faults

Fault	Handler	Bit name	Fault status register
Bus error on a vector read	HardFault	VECTTBL	HardFault Status Register
Fault escalated to a HardFault		FORCED	
MPU or default memory map mismatch:	MemManage		MemManage Fault Status Register
on instruction access		IACCVIOL ¹	
on data access		DACCVIOL	
during exception stacking		MSTKERR	
during exception unstacking		MUNSKERR	

Table 24 • Faults (continued)

Fault	Handler	Bit name	Fault status register
Bus error:	BusFault		
during exception stacking		STKERR	BusFault Status Register
during exception unstacking		UNSTKERR	
during instruction prefetch		IBUSERR	
Precise data bus error		PRECISERR	
Imprecise data bus error		IMPRECISERR	
Attempt to access a coprocessor	UsageFault	NOCP	UsageFault Status Register
Undefined instruction		UNDEFINSTR	
Attempt to enter an invalid instruction set state ²		INVSTATE	
Invalid EXC_RETURN value	UsageFault	INVPC	UsageFault Status Register
Illegal unaligned load or store		UNALIGNED	
Divide By 0		DIVBYZERO	

1. Occurs on an access to an XN region even if the processor does not include an MPU or the MPU is disabled.
2. Attempting to use an instruction set other than the Thumb instruction set or returns to a non load/store-multiple instruction with ICI continuation.

3.5.4.2 Fault Escalation and HardFaults

All faults exceptions except for HardFault have configurable exception priority, see [System Handler Priority Registers](#), page 110. Software can disable execution of the handlers for these faults, see [System Handler Control and State Register](#), page 112.

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler, as described in [Exception Model](#), page 37

In some situations, a fault with configurable priority is treated as a HardFault. This is called *priority escalation*, and the fault is described as *escalated to HardFault*. Escalation to HardFault occurs when:

A fault handler causes the same kind of fault as the one it is servicing. This escalation to HardFault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.

A fault handler causes a fault with the same or lower priority as the fault it is servicing. This is because the handler for the new fault cannot preempt the currently executing fault handler.

An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.

A fault occurs and the handler for that fault is not enabled.

If a BusFault occurs during a stack push when entering a BusFault handler, the BusFault does not escalate to a HardFault. This means that if a corrupted stack causes a fault, the fault handler executes

even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

Only Reset and NMI can preempt the fixed priority HardFault. A HardFault can preempt any exception other than Reset, NMI, or another HardFault.

3.5.4.3 Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For BusFaults and MemManage faults, the fault address register indicates the address accessed by the operation that caused the fault, as detailed in the following table.

Table 25 • Fault Status and Fault Address Registers

Handler	Status Register Name	Address Register Name	Register Description
HardFault	HFSR		HardFault Status Register
MemManage	MMFSR	MMFAR	MemManage Fault Status Register MemManage Fault Address Register
BusFault	BFSR	BFAR	BusFault Status Register BusFault Address Register
UsageFault	UFSR		UsageFault Status Register

3.5.4.4 Lockup

The processor enters a lockup state if a fault occurs when executing the NMI or HardFault handlers. When the processor is in lockup state it does not execute any instructions. The processor remains in lockup state until either:

- It is reset
- An NMI occurs
- It is halted by the debugger.

Note: If lockup state occurs from the NMI handler a subsequent NMI does not cause the processor to leave lockup state.

3.5.5 Power Management

The Cortex-M3 processor sleep modes reduce power consumption:

- Sleep mode stops the processor clock.
- Deep sleep mode stops the system clock and switches off the PLL and flash memory.

The SLEEPDEEP bit of the SCR selects which Sleep mode is used, refer to [System Control Register](#), page 108.

This section describes the mechanisms for entering Sleep mode, and the conditions for waking up from Sleep mode.

3.5.5.1 Entering Sleep Mode

This section describes the mechanisms software can use to put the processor into Sleep mode.

The system can generate spurious wakeup events, for example a debug operation wakes up the processor. Therefore software must be able to put the processor back into Sleep mode after such an event. A program might have an idle loop to put the processor back to Sleep mode.

3.5.5.1.1 Wait for Interrupt

The *wait for interrupt* instruction, WFI, causes immediate entry to sleep mode unless the wake-up condition is true, see [Wakeup from WFI or sleep-on-exit](#), page 46. When the processor executes a WFI

instruction it stops executing instructions and enters sleep mode. See [Wakeup from WFE](#), page 46 for more information.

3.5.5.1.2 Wait for Event

The *wait for event* instruction, WFE, causes entry to sleep mode dependent on the value of a one-bit event register. When the processor executes a WFE instruction, it checks the value of the event register:

0: The processor stops executing instructions and enters Sleep mode.

1: The processor clears the register to 0 and continues executing instructions without entering Sleep mode.

See [WFI](#), page 94 for more information.

If the event register is 1, this indicate that the processor must not enter Sleep mode on execution of a WFE instruction. Typically, this is because an external event signal is asserted, or a processor in the system has executed an SEV instruction, see [SEV](#), page 93. Software cannot access this register directly.

3.5.5.1.3 Sleep-on-exit

If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of all exception handles it returns to Thread mode and immediately enters Sleep mode. Use this mechanism in applications that only require the processor to run when an exception occurs.

3.5.5.2 Wakeup from Sleep Mode

The conditions for the processor to wakeup depend on the mechanism that cause it to enter sleep mode.

3.5.5.2.1 Wakeup from WFI or sleep-on-exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry.

Some embedded systems might have to execute system restore tasks after the processor wakes up, and before it executes an interrupt handler. To achieve this set the PRIMASK bit to 1 and the FAULTMASK bit to 0. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor sets PRIMASK to zero. For more information about PRIMASK and FAULTMASK see [Exception Mask Registers](#), page 25.

3.5.5.2.2 Wakeup from WFE

The processor wakes up if:

- it detects an exception with sufficient priority to cause exception entry
- it detects an external event signal, see [External Event Input](#), page 47
- in a multiprocessor system, another processor in the system executes an SEV instruction

In addition, if the SEVONPEND bit in the SCR is set to 1, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about the SCR see [System Control Register](#), page 108.

3.5.5.3 The Wakeup Interrupt Controller

The Wakeup Interrupt Controller (WIC) is a peripheral that can detect an interrupt and wake the processor from deep sleep mode. The WIC is enabled only when the DEEPSLEEP bit in the SCR is set to 1, see [System Control Register](#), page 108.

The WIC is not programmable, and does not have any registers or user interface. It operates entirely from hardware signals.

When the WIC is enabled and the processor enters deep sleep mode, the power management unit in the system can power down most of the Cortex-M3 processor. This has the side effect of stopping the SysTick timer. When the WIC receives an interrupt, it takes a number of clock cycles to wakeup the processor and restore its state, before it can process the interrupt. This means interrupt latency is increased in deep sleep mode.

Note: If the processor detects a connection to a debugger it disables the WIC.

3.5.5.4 External Event Input

The processor provides an external event input signal. Peripherals can drive this signal, either to wake the processor from WFE, or to set the internal WFE event register to one to indicate that the processor must not enter Sleep mode on a later WFE instruction. See [Wait for Event](#), page 46 for more information.

3.5.5.5 Power Management Programming Hints

ISO/IEC C cannot directly generate the WFI and WFE instructions. The CMSIS provides the following functions for these instructions:

```
void __WFE(void) // Wait for Event
void __WFI(void) // Wait for Interrupt
```

3.6 Cortex-M3 Processor Instruction Set

This section is the reference material for the Cortex-M3 processor instruction set description in this user guide.

3.6.1 Instruction Set Summary

The processor implements a version of the Thumb instruction set. The following table lists the supported instructions.

In the following table:

- angle brackets, <>, enclose alternative forms of the operand
- braces, {}, enclose optional operands
- the Operands column is not exhaustive
- Op2 is a flexible second operand that can be either a register or a constant
- most instructions can use an optional condition code suffix.

For more information on the instructions and operands, see the instruction descriptions.

Table 26 • Cortex-M3 Processor Instructions

Mnemonic	Operands	Brief description	Flags
ADC, ADCS	{Rd,} Rn, Op2	Add with Carry	N, Z, C, V
ADD, ADDS	{Rd,} Rn, Op2	Add	N, Z, C, V
ADD, ADDW	{Rd,} Rn, #imm12	Add	N, Z, C, V
ADR	Rd, label	Load PC-relative Address	
AND, ANDS	{Rd,} Rn, Op2	Logical AND	N, Z, C
ASR, ASRS	Rd, Rm, <Rs n>	Arithmetic Shift Right	N, Z, C
B	label	Branch	
BFC	Rd, #lsb, #width	Bit Field Clear	
BFI	Rd, Rn, #lsb, #width	Bit Field Insert	
BIC, BICS	{Rd,} Rn, Op2	Bit Clear	N, Z, C
BKPT	#imm	Breakpoint	
BL	label	Branch with Link	
BLX	Rm	Branch indirect with Link	
BX	Rm	Branch indirect	
CBNZ	Rn, label	Compare and Branch if Non Zero	
CBZ	Rn, label	Compare and Branch if Zero	
CLREX		Clear Exclusive	

Table 26 • Cortex-M3 Processor Instructions (continued)

Mnemonic	Operands	Brief description	Flags
CLZ	Rd, Rm	Count Leading Zeros	
CMN	Rn, Op2	Compare Negative	N, Z, C, V
CMP	Rn, Op2	Compare	N, Z, C, V
CPSID	i	Change Processor State, Disable Interrupts	
CPSIE	i	Change Processor State, Enable Interrupts	
DMB		Data Memory Barrier	
DSB		Data Synchronization Barrier	
EOR, EORS	{Rd,} Rn, Op2	Exclusive OR	N, Z, C
ISB		Instruction Synchronization Barrier	
IT		If-Then condition block	
LDM	Rn{!}, reglist	Load Multiple registers, increment after	
LDMDB, LDMEA	Rn{!}, reglist	Load Multiple registers, decrement before	
LDMFD, LDMIA	Rn{!}, reglist	Load Multiple registers, increment after	
LDR	Rt, [Rn, #offset]	Load Register with word	
LDRB, LDRBT	Rt, [Rn, #offset]	Load Register with byte	
LDRD	Rt, Rt2, [Rn, #offset]	Load Register with two bytes	
LDREX	Rt, [Rn, #offset]	Load Register Exclusive	
LDREXB	Rt, [Rn]	Load Register Exclusive with Byte	
LDREXH	Rt, [Rn]	Load Register Exclusive with Halfword	
LDRH, LDRHT	Rt, [Rn, #offset]	Load Register with Halfword	
LDRSB, LDRSBT	Rt, [Rn, #offset]	Load Register with Signed Byte	
LDRSH, LDRSHT	Rt, [Rn, #offset]	Load Register with Signed Halfword	
LDRT	Rt, [Rn, #offset]	Load Register with word	
LSL, LSLS	Rd, Rm, <Rs n>	Logical Shift Left	N, Z, C
LSR, LSRS	Rd, Rm, <Rs n>	Logical Shift Right	N, Z, C
MLA	Rd, Rn, Rm, Ra	Multiply with Accumulate, 32-bit result	
MLS	Rd, Rn, Rm, Ra	Multiply and Subtract, 32-bit result	
MOV, MOVS	Rd, Op2	Move	N, Z, C
MOVT	Rd, #imm16	Move Top	
MOVW, MOV	Rd, #imm16	Move 16-bit constant	N, Z, C
MRS	Rd, spec_reg	Move from Special Register to general register	
MSR	spec_reg, Rm	Move from general register to Special Register	N, Z, C, V
MUL, MULS	{Rd,} Rn, Rm	Multiply, 32-bit result	N, Z
MVN, MVNS	Rd, Op2	Move NOT	N, Z, C
NOP		No Operation	

Table 26 • Cortex-M3 Processor Instructions (continued)

Mnemonic	Operands	Brief description	Flags
ORN, ORNS	{Rd,} Rn, Op2	Logical OR NOT	N, Z, C
ORR, ORRS	{Rd,} Rn, Op2	Logical OR	N, Z, C
POP	reglist	Pop registers from stack	
PUSH	reglist	Push registers onto stack	
RBIT	Rd, Rn	Reverse Bits	
REV	Rd, Rn	Reverse byte order in a word	
REV16	Rd, Rn	Reverse byte order in each halfword	
REVSH	Rd, Rn	Reverse byte order in bottom halfword and sign extend	
ROR, RORS	Rd, Rm, <Rs >#n	Rotate Right	N, Z, C
RRX, RRXS	Rd, Rm	Rotate Right with Extend	N, Z, C
RSB, RSBS	{Rd,} Rn, Op2	Reverse Subtract	N, Z, C, V
SBC, SBCS	{Rd,} Rn, Op2	Subtract with Carry	N, Z, C, V
SBFX	Rd, Rn, #lsb, #width	Signed Bit Field Extract	
SDIV	{Rd,} Rn, Rm	Signed Divide	
SEV		Send Event	
SMLAL	RdLo, RdHi, Rn, Rm	Signed Multiply with Accumulate (32 x 32 + 64), 64-bit result	
SMULL	RdLo, RdHi, Rn, Rm	Signed Multiply (32 x 32), 64-bit result	
SSAT	Rd, #n, Rm {,shift #s}	Signed Saturate	Q
STM	Rn{!}, reglist	Store Multiple registers, increment after	
STMDB, STMEA	Rn{!}, reglist	Store Multiple registers, decrement before	
STMFD, STMIA	Rn{!}, reglist	Store Multiple registers, increment after	
STR	Rt, [Rn, #offset]	Store Register word	
STRB, STRBT	Rt, [Rn, #offset]	Store Register byte	
STRD	Rt, Rt2, [Rn, #offset]	Store Register two words	
STREX	Rd, Rt, [Rn, #offset]	Store Register Exclusive	
STREXB	Rd, Rt, [Rn]	Store Register Exclusive Byte	
STREXH	Rd, Rt, [Rn]	Store Register Exclusive Halfword	
STRH, STRHT	Rt, [Rn, #offset]	Store Register Halfword	
STRT	Rt, [Rn, #offset]	Store Register word	
SUB, SUBS	{Rd,} Rn, Op2	Subtract	N, Z, C, V
SUB, SUBW	{Rd,} Rn, #imm12	Subtract	N, Z, C, V
SVC	#imm	Supervisor Call	
SXTB	{Rd,} Rm {,ROR #n}	Sign extend a byte	
SXTH	{Rd,} Rm {,ROR #n}	Sign extend a halfword	
TBB	[Rn, Rm]	Table Branch Byte	

Table 26 • Cortex-M3 Processor Instructions (continued)

Mnemonic	Operands	Brief description	Flags
TBH	[Rn, Rm, LSL #1]	Table Branch Halfword	
TEQ	Rn, Op2	Test Equivalence	N, Z, C
TST	Rn, Op2	Test	N, Z, C
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract	
UDIV	{Rd,} Rn, Rm	Unsigned Divide	
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result	
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result	
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate	Q
UXTB	{Rd,} Rm {,ROR #n}	Zero extend a Byte	
UXTH	{Rd,} Rm {,ROR #n}	Zero extend a Halfword	
WFE		Wait for Event	
WFI		Wait for Interrupt	

3.6.2 CMSIS Functions

ISO/IEC C code cannot directly access some Cortex-M3 processor instructions. This section describes intrinsic functions that can generate these instructions, provided by the CMSIS and that might be provided by a C compiler. If a C compiler does not support an appropriate intrinsic function, you might have to use inline assembler to access some instructions.

The following table lists the intrinsic functions that the CMSIS provides to generate instructions that ISO/IEC C code cannot directly access.

Table 27 • CMSIS Functions to Generate some Cortex-M3 Processor instructions

Instruction	CMSIS function
CPSIE I	void __enable_irq(void)
CPSID I	void __disable_irq(void)
CPSIE F	void __enable_fault_irq(void)
CPSID F	void __disable_fault_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
REV	uint32_t __REV(uint32_t int value)
REV16	uint32_t __REV16(uint32_t int value)
REVSH	uint32_t __REVSH(uint32_t int value)
RBIT	uint32_t __RBIT(uint32_t int value)
SEV	void __SEV(void)
WFE	void __WFE(void)
WFI	void __WFI(void)

The following table lists the functions that CMSIS provides for accessing the special registers using MRS and MSR instructions.

Table 28 • CMSIS Functions to Access the Special Registers

Special Register	Access	CMSIS function
PRIMASK	Read	uint32_t __get_PRIMASK (void)
	Write	void __set_PRIMASK (uint32_t value)
FAULTMASK	Read	uint32_t __get_FAULTMASK (void)
	Write	void __set_FAULTMASK (uint32_t value)
BASEPRI	Read	uint32_t __get_BASEPRI (void)
	Write	void __set_BASEPRI (uint32_t value)
CONTROL	Read	uint32_t __get_CONTROL (void)
	Write	void __set_CONTROL (uint32_t value)
MSP	Read	uint32_t __get_MSP (void)
	Write	void __set_MSP (uint32_t TopOfMainStack)
PSP	Read	uint32_t __get_PSP (void)
	Write	void __set_PSP (uint32_t TopOfProcStack)

3.6.3 About the Instruction Descriptions

The following sections provide more information about using the instructions:

3.6.3.1 Operands

An instruction operand can be an ARM register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. When there is a destination register in the instruction, it is usually specified before the operands.

Operands in some instructions are flexible in that they can either be a register or a constant. Refer to [Flexible Second Operand](#), page 51.

3.6.3.2 Restrictions when Using PC or SP

Many instructions have restrictions on whether you can use the *Program Counter* (PC) or *Stack Pointer* (SP) for the operands or destination register. See instruction descriptions for more information.

Bit[0] of any address you write to the PC with a BX, BLX, LDM, LDR, or POP instruction must be 1 for correct execution, because this bit indicates the required instruction set, and the Cortex-M3 processor only supports Thumb instructions.

3.6.3.3 Flexible Second Operand

Many general data processing instructions have a flexible second operand. This is shown as *Operand2* in the descriptions of the syntax of each instruction.

Operand2 can be a constant or a register with optional shift.

3.6.3.3.1 Constant

You specify an *Operand2* constant in the form:

#constant

where *constant* can be:

- any constant that can be produced by shifting an 8-bit value left by any number of bits within a 32-bit word
- any constant of the form 0x00XY00XY

- any constant of the form 0xXY00XY00
- any constant of the form 0xXYXYXYXY

In the constants shown above, X and Y are hexadecimal digits.

In addition, in a small number of instructions, *constant* can take a wider range of values. These are described in the individual instruction descriptions.

When an Operand2 constant is used with the instructions MOVs, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to bit[31] of the constant, if the constant is greater than 255 and can be produced by shifting an 8-bit value. These instructions do not affect the carry flag if Operand2 is any other constant.

Instruction substitution

Your assembler might be able to produce an equivalent instruction in cases where you specify a constant that is not permitted. For example, an assembler might assemble the instruction `CMP Rd, #0xFFFFFFFFE` as the equivalent instruction `CMN Rd, #0x2`.

3.6.3.3.2 Register with Optional Shift

You specify an Operand2 register in the form:

Rm {, *shift*}

where:

Rm: is the register holding the data for the second operand.

shift: is an optional shift to be applied to *Rm*. It can be one of:

ASR #*n*: arithmetic shift right *n* bits, $1 \leq n \leq 32$.

LSL #*n*: logical shift left *n* bits, $1 \leq n \leq 31$.

LSR #*n*: logical shift right *n* bits, $1 \leq n \leq 32$.

ROR #*n*: rotate right *n* bits, $1 \leq n \leq 31$.

RRX: rotate right one bit, with extend.

-: if omitted, no shift occurs, equivalent to LSL #0.

If you omit the shift, or specify LSL #0, the instruction uses the value in *Rm*.

If you specify a shift, the shift is applied to the value in *Rm*, and the resulting 32-bit value is used by the instruction. However, the contents in the register *Rm* remains unchanged. Specifying a register with shift also updates the carry flag when used with certain instructions. For information on the shift operations and how they affect the carry flag, refer to [Shift Operations](#), page 52.

3.6.3.4 Shift Operations

Register shift operations move the bits in a register left or right by a specified number of bits, the *shift length*. Register shift can be performed:

- directly by the instructions ASR, LSR, LSL, ROR, and RRX, and the result is written to a destination register
- during the calculation of *Operand2* by the instructions that specify the second operand as a register with shift, refer to [Flexible Second Operand](#), page 51. The result is used by the instruction.

The permitted shift lengths depend on the shift type and the instruction, refer to the individual instruction description or [Flexible Second Operand](#), page 51. If the shift length is 0, no shift occurs. Register shift operations update the carry flag except when the specified shift length is 0. The following sub-sections describe the various shift operations and how they affect the carry flag. In these descriptions, *Rm* is the register containing the value to be shifted, and *n* is the shift length.

3.6.3.4.1 ASR

Arithmetic shift right by *n* bits moves the left-hand 32-*n* bits of the register *Rm*, to the right by *n* places, into the right-hand 32-*n* bits of the result. And it copies the original bit[31] of the register into the left-hand *n* bits of the result. Refer to [Figure 18](#), page 53.

You can use the ASR $\#n$ operation to divide the value in the register Rm by 2^n , with the result being rounded towards negative-infinity.

When the instruction is ASRS or when ASR $\#n$ is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[$n-1$], of the register Rm .

- If n is 32 or more, then all the bits in the result are set to the value of bit[31] of Rm .
- If n is 32 or more and the carry flag is updated, it is updated to the value of bit[31] of Rm .

Figure 18 • ASR#3



3.6.3.4.2 LSR

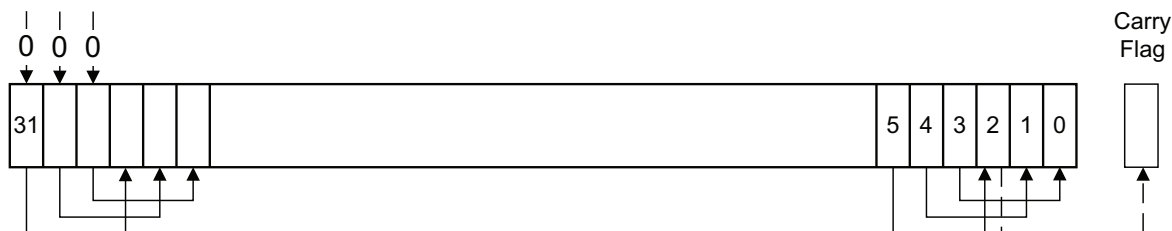
Logical shift right by n bits moves the left-hand $32-n$ bits of the register Rm , to the right by n places, into the right-hand $32-n$ bits of the result. And it sets the left-hand n bits of the result to 0. See the following figure.

You can use the LSR $\#n$ operation to divide the value in the register Rm by 2^n , if the value is regarded as an unsigned integer.

When the instruction is LSRS or when LSR $\#n$ is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[$n-1$], of the register Rm .

- If n is 32 or more, then all the bits in the result are cleared to 0.
- If n is 33 or more and the carry flag is updated, it is updated to 0.

Figure 19 • LSR



3.6.3.4.3 LSL

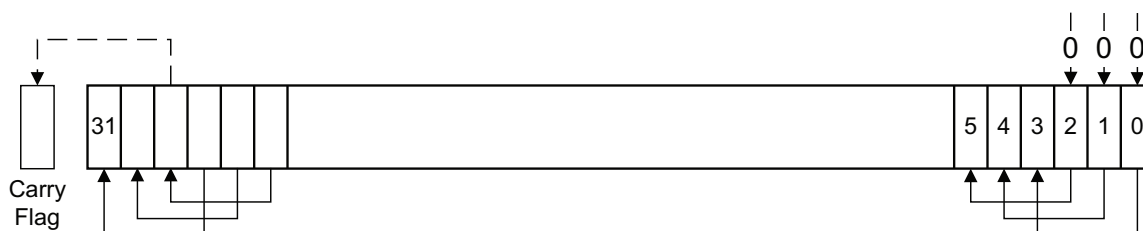
Logical shift left by n bits moves the right-hand $32-n$ bits of the register Rm , to the left by n places, into the left-hand $32-n$ bits of the result. And it sets the right-hand n bits of the result to 0. See Figure 20, page 54.

You can use the LSL $\#n$ operation to multiply the value in the register Rm by 2^n , if the value is regarded as an unsigned integer or a two's complement signed integer. Overflow can occur without warning.

When the instruction is LSLS or when LSL $\#n$, with non-zero n , is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit shifted out, bit[32- n], of the register Rm . These instructions do not affect the carry flag when used with LSL $\#0$.

- If n is 32 or more, then all the bits in the result are cleared to 0.
- If n is 33 or more and the carry flag is updated, it is updated to 0.

Figure 20 • LSL



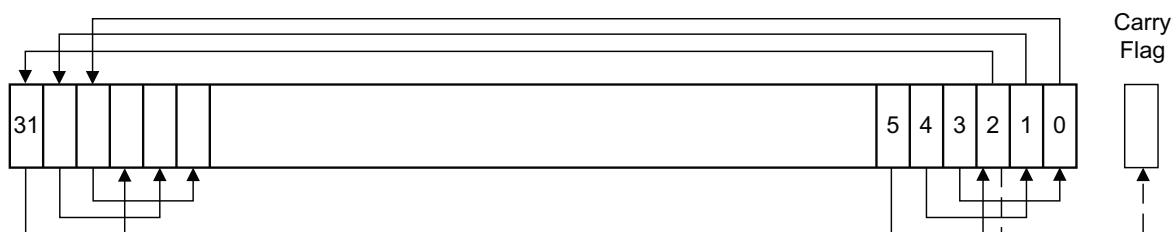
3.6.3.4.4 ROR

Rotate right by n bits moves the left-hand $32-n$ bits of the register Rm , to the right by n places, into the right-hand $32-n$ bits of the result. And it moves the right-hand n bits of the register into the left-hand n bits of the result. See the following figure.

When the instruction is RORS or when ROR $\#n$ is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to the last bit rotation, bit[$n-1$], of the register Rm .

- If n is 32, then the value of the result is same as the value in Rm , and if the carry flag is updated, it is updated to bit[31] of Rm .
- ROR with shift length, n , more than 32 is the same as ROR with shift length $n-32$.

Figure 21 • ROR

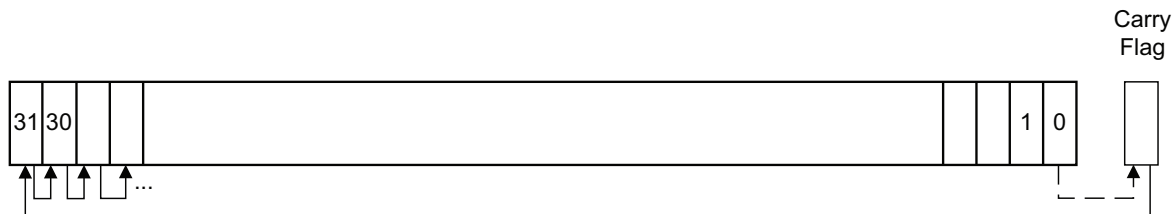


3.6.3.4.5 RRX

Rotate right with extend moves the bits of the register *Rm* to the right by one bit. And it copies the carry flag into bit[31] of the result. See the following figure.

When the instruction is RRXS or when RRX is used in *Operand2* with the instructions MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ or TST, the carry flag is updated to bit[0] of the register *Rm*.

Figure 22 • RRX



3.6.3.5 Address Alignment

An aligned access is an operation where a word-aligned address is used for a word, dual word, or multiple word access, or where a halfword-aligned address is used for a halfword access. Byte accesses are always aligned.

The Cortex-M3 processor supports unaligned access only for the following instructions:

- LDR, LDRT
- LDRH, LDRHT
- LDRSH, LDRSHT
- STR, STRT
- STRH, STRHT

All other load and store instructions generate a UsageFault exception if they perform an unaligned access, and therefore their accesses must be address aligned. For more information about UsageFaults refer to [Fault Handling](#), page 43.

Unaligned accesses are usually slower than aligned accesses. In addition, some memory regions might not support unaligned accesses. Therefore, ARM recommends that programmers ensure that accesses are aligned. To trap accidental generation of unaligned accesses, use the UNALIGN_TRP bit in the Configuration and Control Register, refer to [Configuration and Control Register](#), page 109.

3.6.3.6 PC-relative Expressions

A PC-relative expression or *label* is a symbol that represents the address of an instruction or literal data. It is represented in the instruction as the PC value plus or minus a numeric offset. The assembler calculates the required offset from the label and the address of the current instruction. If the offset is too big, the assembler produces an error.

- For B, BL, CBNZ, and CBZ instructions, the value of the PC is the address of the current instruction plus 4 bytes.
- For most other instructions that use labels, the value of the PC is the address of the current instruction plus 4 bytes, with bit[1] of the result cleared to 0 to make it word-aligned.
- Your assembler might permit other syntaxes for PC-relative expressions, such as a label plus or minus a number, or an expression of the form [PC, #number].

3.6.3.7 Conditional Execution

Most data processing instructions can optionally update the condition flags in the *Application Program Status Register* (APSR) according to the result of the operation; see [Application Program Status Register](#), page 23. Some instructions update all flags, and some only update a subset. If a flag is not updated, the original value is preserved. See the instruction descriptions for the flags they affect.

You can execute an instruction conditionally, based on the condition flags set in another instruction, either:

- Immediately after the instruction that updated the flags
- After any number of intervening instructions that have not updated the flags.

Conditional execution is available by using conditional branches or by adding condition code suffixes to instructions. See [Table 29](#), page 57 for a list of the suffixes to add to instructions to make them conditional instructions. The condition code suffix enables the processor to test a condition based on the flags. If the condition test of a conditional instruction fails, the instruction:

- Does not execute
- Does not write any value to its destination register
- Does not affect any of the flags
- Does not generate any exception.

Conditional instructions, except for conditional branches, must be inside an If-Then instruction block. See [IT](#), page 85 for more information and restrictions when using the IT instruction. Depending on the vendor, the assembler might automatically insert an IT instruction if you have conditional instructions outside the IT block.

Use the CBZ and CBNZ instructions to compare the value of a register against zero and branch on the result.

This section describes the condition flags and condition code suffixes.

3.6.3.7.1 Condition Flags

The APSR contains the following condition flags:

N: Set to 1 when the result of the operation was negative, cleared to 0 otherwise.

Z: Set to 1 when the result of the operation was zero, cleared to 0 otherwise.

C: Set to 1 when the operation resulted in a carry, cleared to 0 otherwise.

V: Set to 1 when the operation caused overflow, cleared to 0 otherwise.

For more information about the APSR, refer to [Program Status Register](#), page 22.

A carry occurs:

- If the result of an addition is greater than or equal to 2^{32}
- If the result of a subtraction is positive or zero
- As the result of an inline barrel shifter operation in a move or logical instruction

Overflow occurs when the sign of the result, in bit[31], does not match the sign of the result had the operation been performed at infinite precision, for example:

- If adding two negative values results in a positive value
- If adding two positive values results in a negative value
- If subtracting a positive value from a negative value generates a positive value
- If subtracting a negative value from a positive value generates a negative value

The Compare operations are identical to subtracting, for CMP, or adding, for CMN, except that the result is discarded. See the instruction descriptions for more information. Most instructions update the status flags only if the S suffix is specified. See the instruction descriptions for more information.

3.6.3.7.2 Condition Code Suffixes

The instructions that can be conditional have an optional condition code, shown in syntax descriptions as *{cond}*. Conditional execution requires a preceding IT instruction. An instruction with a condition code is only executed if the condition code flags in the APSR meet the specified condition. The following table shows the condition codes to use. You can use conditional execution with the IT instruction to reduce the

number of branch instructions in code. The table also shows the relationship between condition code suffixes and the N, Z, C, and V flags.

Table 29 • Condition Code Suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned
CC or LO	C = 0	Lower, unsigned
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned
LS	C = 0 or Z = 1	Lower or same, unsigned
GE	N = V	Greater than or equal, signed
LT	N != V	Less than, signed
GT	Z = 0 and N = V	Greater than, signed
LE	Z = 1 and N != V	Less than or equal, signed
AL	Can have any value	Always. This is the default when no suffix is specified.

The following example shows the use of a conditional instruction to find the absolute value of a number. R0 = abs(R1).

Example 1

Absolute value

```

MOVS    R0, R1          ; R0 = R1, setting flags
IT      MI              ; skipping next instruction if value 0 or positive
RSBMI   R0, R0, #0      ; If negative, R0 = -R0

```

The following example shows the use of conditional instructions to update the value of R4 if the signed values R0 is greater than R1 and R2 is greater than R3.

Example 2

Compare and update value

```

CMP     R0, R1          ; Compare R0 and R1, setting flags
ITT     GT              ; Skip next two instructions unless GT condition holds
CMPGT   R2, R3          ; If 'greater than', compare R2 and R3, setting flags
MOVGT   R4, R5          ; If still 'greater than', do R4 = R5

```

3.6.3.8 Instruction Width Selection

There are many instructions that can generate either a 16-bit encoding or a 32-bit encoding depending on the operands and destination register specified. For some of these instructions, you can force a specific instruction size by using an instruction width suffix. The `.W` suffix forces a 32-bit instruction encoding. The `.N` suffix forces a 16-bit instruction encoding.

If you specify an instruction width suffix and the assembler cannot generate an instruction encoding of the requested width, it generates an error.

Note: In some cases it might be necessary to specify the `.W` suffix, for example if the operand is the label of an instruction or literal data, as in the case of branch instructions. This is because the assembler might not automatically generate the right size encoding.

To use an instruction width suffix, place it immediately after the instruction mnemonic and condition code, if any. The following example shows instructions with the instruction width suffix.

Example 3

Instruction width selection

```
BCS.W label      ; creates a 32-bit instruction even for a short branch
ADDS.W R0, R0, R1 ; creates a 32-bit instruction even though the same
                  ; operation can be done by a 16-bit instruction
```

3.6.4 Memory Access Instructions

The following table provides memory access instructions:

Table 30 • Memory Access Instructions

Mnemonic	Brief Description	See
ADR	Generate PC-relative address	ADR , page 59
CLREX	Clear Exclusive	CLREX , page 67
LDM{mode}	Load Multiple registers	LDM and STM , page 64
LDR{type}	Load Register using immediate offset	LDR and STR, Immediate Offset , page 59
LDR{type}	Load Register using register offset	LDR and STR, Register Offset , page 61
LDR{type}T	Load Register with unprivileged access	LDR and STR, Unprivileged , page 62
LDR	Load Register using PC-relative address	LDR, PC-relative , page 63
LDREX{type}	Load Register Exclusive	LDREX and STREX , page 66
POP	Pop registers from stack	PUSH and POP , page 65
PUSH	Push registers onto stack	PUSH and POP , page 65
STM{mode}	Store Multiple registers	LDM and STM , page 64
STR{type}	Store Register using immediate offset	LDR and STR, Immediate Offset , page 59
STR{type}	Store Register using register offset	LDR and STR, Register Offset , page 61
STR{type}T	Store Register with unprivileged access	LDR and STR, Unprivileged , page 62
STREX{type}	Store Register Exclusive	LDREX and STREX , page 66

3.6.4.1 ADR

Generate PC-relative address.

3.6.4.1.1 Syntax

```
ADR{cond} Rd, label
```

where:

- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *Rd* is the destination register
- *label* is a PC-relative expression. See [PC-relative Expressions](#), page 55.

3.6.4.1.2 Operation

ADR generates an address by adding an immediate value to the PC, and writes the result to the destination register.

ADR provides the means by which position-independent code can be generated, because the address is PC-relative.

If you use ADR to generate a target address for a BX or BLX instruction, you must ensure that bit[0] of the address you generate is set to 1 for correct execution.

Values of *label* must be within the range of -4095 to +4095 from the address in the PC.

Note: You may have to use the *.W* suffix to get the maximum offset range or to generate addresses that are not word-aligned. See [Instruction Width Selection](#), page 58.

3.6.4.1.3 Restrictions

Rd must not be SP and must not be PC.

3.6.4.1.4 Condition flags

This instruction does not change the flags.

Examples

```
ADR    R1, TextMessage    ; Write address value of a location labelled as
                                ; TextMessage to R1
```

3.6.4.2 LDR and STR, Immediate Offset

Load and Store with immediate offset, pre-indexed immediate offset, or post-indexed immediate offset.

3.6.4.2.1 Syntax

```
op{type}{cond} Rt, [Rn, #offset]    ; immediate offset
op{type}{cond} Rt, [Rn, #offset]!   ; pre-indexed
op{type}{cond} Rt, [Rn], #offset    ; post-indexed
opD{cond} Rt, Rt2, [Rn, #offset]    ; immediate offset, two words
opD{cond} Rt, Rt2, [Rn, #offset]!   ; pre-indexed, two words
opD{cond} Rt, Rt2, [Rn], #offset    ; post-indexed, two words
```

where:

- *op* is either LDR (load register) or STR (store register)
- *type* is one of:
 - B: unsigned byte, zero extend to 32 bits on loads.
 - SB: signed byte, sign extend to 32 bits (LDR only).
 - H: unsigned halfword, zero extend to 32 bits on loads.
 - SH: signed halfword, sign extend to 32 bits (LDR only).
 - -: omit, for word.
- *cond* is an optional condition code; see [Conditional Execution](#), page 55.

- Rt is the register to load or store.
- Rn is the register on which the memory address is based.
- $offset$ is an offset from Rn . If $offset$ is omitted, the address is the contents of Rn .
- $Rt2$ is the additional register to load or store for two-word operations.

3.6.4.2.2 Operation

LDR instructions load one or two registers with a value from memory.

STR instructions store one or two register values to memory.

Load and store instructions with immediate offset can use the following addressing modes:

Offset Addressing

The offset value is added to or subtracted from the address obtained from the register Rn . The result is used as the address for the memory access. The register Rn is unaltered. The assembly language syntax for this mode is:

$[Rn, \#offset]$

Pre-indexed addressing

The offset value is added to or subtracted from the address obtained from the register Rn . The result is used as the address for the memory access and written back into the register Rn . The assembly language syntax for this mode is:

$[Rn, \#offset]!$

Post-indexed addressing

The address obtained from the register Rn is used as the address for the memory access. The offset value is added to or subtracted from the address, and written back into the register Rn . The assembly language syntax for this mode is:

$[Rn], \#offset$

The value to load or store can be a byte, halfword, word, or two words. Bytes and halfwords can either be signed or unsigned. Refer to [Address Alignment](#), page 55.

The following table lists the ranges of offset for immediate, pre-indexed and post-indexed forms.

Table 31 • Offset Ranges

Instruction Type	Immediate Offset	Pre-Indexed	Post-Indexed
Word, halfword, signed halfword, byte, or signed byte	-255 to 4095	-255 to 255	-255 to 255
Two words	Multiple of 4 in the range -1020 to 1020	Multiple of 4 in the range -1020 to 1020	Multiple of 4 in the range -1020 to 1020

3.6.4.2.3 Restrictions

For load instructions:

- Rt can be SP or PC for word loads only
- Rt must be different from $Rt2$ for two-word loads
- Rn must be different from Rt and $Rt2$ in the pre-indexed or post-indexed forms.

When Rt is PC in a word load instruction:

- bit[0] of the loaded value must be 1 for correct execution
- a branch occurs to the address created by changing bit[0] of the loaded value to 0
- if the instruction is conditional, it must be the last instruction in the IT block.

For store instructions:

- Rt can be SP for word stores only
- Rt must not be PC

- *Rn* must not be PC
- *Rn* must be different from *Rt* and *Rt2* in the pre-indexed or post-indexed forms.

3.6.4.2.4 Condition flags

These instructions do not change the flags.

Examples

```
LDR      R8, [R10]                ; Loads R8 from the address in R10.

LDRNE    R2, [R5, #960]!          ; Loads (conditionally) R2 from a word
                                   ; 960 bytes above the address in R5, and
                                   ; increments R5 by 960

STR      R2, [R9, #const-struct] ; const-struct is an expression evaluating
                                   ; to a constant in the range 0-4095.

STRH     R3, [R4], #4             ; Store R3 as halfword data into address in
                                   ; R4, then increment R4 by 4

LDRD     R8, R9, [R3, #0x20]      ; Load R8 from a word 8 bytes above the
                                   ; address in R3, and load R9 from a word 9
                                   ; bytes above the address in R3

STRD     R0, R1, [R8], #-16       ; Store R0 to address in R8, and store R1 to
                                   ; a word 4 bytes above the address in R8,
                                   ; and then decrement R8 by 16.
```

3.6.4.3 LDR and STR, Register Offset

Load and Store with register offset.

3.6.4.3.1 Syntax

```
op{type}{cond} Rt, [Rn, Rm {, LSL #n}]
```

where:

- *op* is either LDR (load register) or STR (store register)
- *type* is one of:
 - B: unsigned byte, zero extend to 32 bits on loads.
 - SB: signed byte, sign extend to 32 bits (LDR only).
 - H: unsigned halfword, zero extend to 32 bits on loads.
 - SH: signed halfword, sign extend to 32 bits (LDR only).
 - -: omit, for word.
- *cond* is an optional condition code, refer to [Conditional Execution](#), page 55.
- *Rt* is the register to load or store.
- *Rn* is the register on which the memory address is based.
- *Rm* is a register containing a value to be used as the offset.
- *LSL #n* is an optional shift, with *n* in the range 0 to 3.

3.6.4.3.2 Operation

LDR instructions load a register with a value from memory.

STR instructions store a register value into memory.

The memory address to load from or store to is at an offset from the register *Rn*. The offset is specified by the register *Rm* and can be shifted left by up to 3 bits using LSL.

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See [Address Alignment](#), page 55.

3.6.4.3.3 Restrictions

In these instructions:

- *Rn* must not be PC
- *Rm* must not be SP and must not be PC
- *Rt* can be SP only for word loads and word stores
- *Rt* can be PC only for word loads.

When *Rt* is PC in a word load instruction:

- Bit[0] of the loaded value must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- If the instruction is conditional, it must be the last instruction in the IT block.

3.6.4.3.4 Condition Flags

These instructions do not change the flags.

Examples

```
STR    R0, [R5, R1]           ; Store value of R0 into an address equal to
                                ; sum of R5 and R1

LDRSB  R0, [R5, R1, LSL #1] ; Read byte value from an address equal to
                                ; sum of R5 and two times R1, sign extended it
                                ; to a word value and put it in R0

STR    R0, [R1, R2, LSL #2] ; Stores R0 to an address equal to sum of R1
                                ; and four times R2.
```

3.6.4.4 LDR and STR, Unprivileged

Load and Store with unprivileged access.

3.6.4.4.1 Syntax

```
op{type}T{cond} Rt, [Rn {, #offset}] ; immediate offset
```

where:

- *op* is either LDR (load register) or STR (store register)
- *type* is one of:
 - B: unsigned byte, zero extend to 32 bits on loads.
 - SB: signed byte, sign extend to 32 bits (LDR only).
 - H: unsigned halfword, zero extend to 32 bits on loads.
 - SH: signed halfword, sign extend to 32 bits (LDR only).
 - -: omit, for word.
- *cond* is an optional condition code, refer to [Conditional Execution](#), page 55.
- *Rt* is the register to load or store.
- *Rn* is the register on which the memory address is based.
- *offset* is an offset from *Rn* and can be 0 to 255. If offset is omitted, the address is the value in *Rn*.

3.6.4.4.2 Operation

These load and store instructions perform the same function as the memory access instructions with immediate offset, refer to [LDR and STR, Immediate Offset](#), page 594. The difference is that these instructions have only unprivileged access even when used in privileged software.

When used in unprivileged software, these instructions behave in exactly the same way as normal memory access instructions with immediate offset.

3.6.4.4.3 Restrictions

In these instructions:

- *Rn* must not be PC.
- *Rt* must not be SP and must not be PC.

3.6.4.4.4 Condition Flags

These instructions do not change the flags.

Examples

```
STRBTEQ R4, [R7]           ; Conditionally store least significant byte in
                             ; R4 to an address in R7, with unprivileged access

LDRHT    R2, [R2, #8]      ; Load halfword value from an address equal to
                             ; sum of R2 and 8 into R2, with unprivileged access.
```

3.6.4.5 LDR, PC-relative

Load register from memory.

3.6.4.5.1 Syntax

```
LDR{type}{cond} Rt, label
LDRD{cond} Rt, Rt2, label      ; Load two words
```

where:

- *type* is one of:
 - B: unsigned byte, zero extend to 32 bits.
 - SB: signed byte, sign extend to 32 bits.
 - H: unsigned halfword, zero extend to 32 bits.
 - SH: signed halfword, sign extend to 32 bits.
 - -: omit, for word.
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *Rt* is the register to load or store.
- *Rt2* is the second register to load or store.
- *label* is a PC-relative expression. See [PC-relative Expressions](#), page 55.

3.6.4.5.2 Operation

LDR loads a register with a value from a PC-relative memory address. The memory address is specified by a label or by an offset from the PC.

The value to load or store can be a byte, halfword, or word. For load instructions, bytes and halfwords can either be signed or unsigned. See [Address Alignment](#), page 55.

label must be within a limited range of the current instruction. The following table shows the possible offsets between *label* and the PC.

Table 32 • Offset Ranges

Instruction type	Offset range
Word, halfword, signed halfword, byte, signed byte	-4095 to 4095
Two words	-1020 to 1020

Note: You might have to use the *.W* suffix to get the maximum offset range. See [Instruction Width Selection](#), page 58.

3.6.4.5.3 Restrictions

In these instructions:

- *Rt* can be SP or PC only for word loads
- *Rt2* must not be SP and must not be PC
- *Rt* must be different from *Rt2*.

When *Rt* is PC in a word load instruction:

- bit[0] of the loaded value must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- if the instruction is conditional, it must be the last instruction in the IT block.

3.6.4.5.4 Condition Flags

These instructions do not change the flags.

Examples

```
LDR      R0, LookUpTable    ; Load R0 with a word of data from an address
                                ; labelled as LookUpTable

LDRSB    R7, localdata      ; Load a byte value from an address labelled
                                ; as localdata, sign extend it to a word
                                ; value, and put it in R7.
```

3.6.4.6 LDM and STM

Load and Store Multiple registers.

3.6.4.6.1 Syntax

```
op{addr_mode}{cond} Rn{!}, reglist
```

where:

- *op* is one of:
 - LDM (Load Multiple registers)
 - STM (Store Multiple registers)
- *addr_mode* is one of:
 - IA (Increment address After each access.) This is the default.
 - DB (Decrement address Before each access.)
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *Rn* is the register on which the memory addresses are based.
- *!* is an optional writeback suffix. If *!* is present the final address, that is loaded from or stored to, is written back into *Rn*.
- *reglist* is a list of one or more registers to be loaded or stored, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range.

LDM and LDMFD are synonyms for LDMIA. LDMFD refers to its use for popping data from Full Descending stacks.

LDMEA is a synonym for LDMDB, and refers to its use for popping data from Empty Ascending stacks.

STM and STMEA are synonyms for STMIA. STMEA refers to its use for pushing data onto Empty Ascending stacks.

STMFD is a synonym for STMDB, and refers to its use for pushing data onto Full Descending stacks.

3.6.4.6.2 Operation

LDM instructions load the registers in *reglist* with word values from memory addresses based on *Rn*.

STM instructions store the word values in the registers in *reglist* to memory addresses based on *Rn*.

For LDM, LDMIA, LDMFD, STM, STMIA, and STMEA the memory addresses used for the accesses are at 4-byte intervals ranging from *Rn* to *Rn* + 4 * (*n*-1), where *n* is the number of registers in *reglist*. The accesses

happens in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest number register using the highest memory address. If the writeback suffix is specified, the value of $Rn + 4 * (n-1)$ is written back to Rn .

For LDMDB, LDMEA, STMDB, and STMFD the memory addresses used for the accesses are at 4-byte intervals ranging from Rn to $Rn - 4 * (n-1)$, where n is the number of registers in *reglist*. The accesses happen in order of decreasing register numbers, with the highest numbered register using the highest memory address and the lowest number register using the lowest memory address. If the writeback suffix is specified, the value of $Rn - 4 * (n-1)$ is written back to Rn .

The PUSH and POP instructions can be expressed in this form. See [PUSH and POP](#), page 65 for details.

3.6.4.6.3 Restrictions

In these instructions:

- Rn must not be PC
- *reglist* must not contain SP
- in any STM instruction, *reglist* must not contain PC
- in any LDM instruction, *reglist* must not contain PC if it contains LR
- *reglist* must not contain Rn if you specify the writeback suffix.

When PC is in *reglist* in an LDM instruction:

- bit[0] of the value loaded to the PC must be 1 for correct execution, and a branch occurs to this halfword-aligned address
- if the instruction is conditional, it must be the last instruction in the IT block.

3.6.4.6.4 Condition Flags

These instructions do not change the flags.

Examples

```
LDM      R8, {R0,R2,R9}      ; LDMIA is a synonym for LDM
```

```
TMDB     R1!, {R3-R6,R11,R12}
```

Incorrect Examples

```
STM      R5!, {R5,R4,R9} ; Value stored for R5 is unpredictable
```

```
LDM      R2, {}             ; There must be at least one register in the list.
```

3.6.4.7 PUSH and POP

Push registers onto, and pop registers off a full-descending stack.

3.6.4.7.1 Syntax

```
PUSH{cond} reglist
```

```
POP{cond} reglist
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

reglist is a non-empty list of registers, enclosed in braces. It can contain register ranges. It must be comma separated if it contains more than one register or register range.

PUSH and POP are synonyms for STMDB and LDM (or LDMIA) with the memory addresses for the access based on SP, and with the final address for the access written back to the SP. PUSH and POP are the preferred mnemonics in these cases.

3.6.4.7.2 Operation

PUSH stores registers on the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

POP loads registers from the stack, with the lowest numbered register using the lowest memory address and the highest numbered register using the highest memory address.

PUSH uses the value in the SP register minus four as the highest memory address, POP uses the value in the SP register as the lowest memory address, implementing a full-descending stack. On completion, PUSH updates the SP register to point to the location of the lowest stored value, POP updates the SP register to point to the location immediately above the highest location loaded.

If a POP instruction includes PC in its *reglist*, a branch to this location is performed when the POP instruction has completed. Bit[0] of the value read for the PC is used to update the APSR T-bit. This bit must be 1 to ensure correct operation.

See [LDM and STM](#), page 64 for more information.

3.6.4.7.3 Restrictions

In these instructions:

- *reglist* must not contain SP
- for the PUSH instruction, *reglist* must not contain PC
- for the POP instruction, *reglist* must not contain PC if it contains LR.

When PC is in *reglist* in a POP instruction:

- bit[0] of the value loaded for PC must be 1 for correct execution
- if the instruction is conditional, it must be the last instruction in the IT block.

3.6.4.7.4 Condition Flags

These instructions do not change the flags.

Examples

```
PUSH {R0,R4-R7} ; Push R0,R4,R5,R6,R7 onto the stack
```

```
PUSH {R2,LR} ; Push R2 and the link-register onto the stack
```

```
POP {R0,R6,PC} ; Pop r0,r6 and PC from the stack, then branch to the new PC
```

3.6.4.8 LDREX and STREX

Load and Store Register Exclusive.

3.6.4.8.1 Syntax

```
LDREX{cond} Rt, [Rn {, #offset}]
```

```
STREX{cond} Rd, Rt, [Rn {, #offset}]
```

```
LDREXB{cond} Rt, [Rn]
```

```
STREXB{cond} Rd, Rt, [Rn]
```

```
LDREXH{cond} Rt, [Rn]
```

```
STREXH{cond} Rd, Rt, [Rn]
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register for the returned status.

Rt is the register to load or store.

Rn is the register on which the memory address is based.

offset is an optional offset applied to the value in *Rn*. If *offset* is omitted, the address is the value in *Rn*.

3.6.4.8.2 Operation

LDREX, LDREXB, and LDREXH load a word, byte, and halfword respectively from a memory address.

STREX, STREXB, and STREXH attempt to store a word, byte, and halfword respectively to a memory address. The address used in any Store-Exclusive instruction must be the same as the address in the most recently executed Load-exclusive instruction. The value stored by the Store-Exclusive instruction

must also have the same data size as the value loaded by the preceding Load-exclusive instruction. This means software must always use a Load-exclusive instruction and a matching Store-Exclusive instruction to perform a synchronization operation, see [Synchronization Primitives](#), page 35.

If an Store-Exclusive instruction performs the store, it writes 0 to its destination register. If it does not perform the store, it writes 1 to its destination register. If the Store-Exclusive instruction writes 0 to the destination register, it is guaranteed that no other process in the system has accessed the memory location between the Load-exclusive and Store-Exclusive instructions.

For reasons of performance, keep the number of instructions between corresponding Load-Exclusive and Store-Exclusive instruction to a minimum.

Note: The result of executing a Store-Exclusive instruction to an address that is different from that used in the preceding Load-Exclusive instruction is unpredictable.

3.6.4.8.3 Restrictions

In these instructions:

- do not use PC
- do not use SP for *Rd* and *Rt*
- for STREX, *Rd* must be different from both *Rt* and *Rn*
- the value of *offset* must be a multiple of four in the range 0-1020.

3.6.4.8.4 Condition Flags

These instructions do not change the flags.

Examples

```

MOV      R1, #0x1           ; Initialize the 'lock taken' value
try
    LDREX  R0, [LockAddr]    ; Load the lock value
    CMP    R0, #0            ; Is the lock free?
    ITT    EQ                ; IT instruction for STREXEQ and CMPEQ
    STREXEQ R0, R1, [LockAddr] ; Try and claim the lock
    CMPEQ  R0, #0            ; Did this succeed?
    BNE    try               ; No - try again
    ....                    ; Yes - we have the lock.
```

3.6.4.9 CLREX

Clear Exclusive.

3.6.4.9.1 Syntax

CLREX{cond}

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.4.9.2 Operation

Use CLREX to make the next STREX, STREXB, or STREXH instruction write 1 to its destination register and fail to perform the store. It is useful in exception handler code to force the failure of the store exclusive if the exception occurs between a load exclusive instruction and the matching store exclusive instruction in a synchronization operation.

See [Synchronization Primitives](#), page 35 for more information.

3.6.4.9.3 Condition Flags

These instructions do not change the flags.

Examples

CLREX

3.6.5 General Data processing instructions

The following table shows the data processing instructions:

Table 33 • Data Processing Instructions

Mnemonic	Brief Description	See
ADC	Add with Carry	"ADD, ADC, SUB, SBC, and RSB" section
ADD	Add	ADD, ADC, SUB, SBC, and RSB, page 69
ADDW	Add	ADD, ADC, SUB, SBC, and RSB, page 69
AND	Logical AND	AND, ORR, EOR, BIC, and ORN, page 70
ASR	Arithmetic Shift Right	ASR, LSL, LSR, ROR, and RRX, page 71
BIC	Bit Clear	AND, ORR, EOR, BIC, and ORN, page 70
CLZ	Count leading zeros	CLZ, page 72
CMN	Compare Negative	CMP and CMN, page 73
CMP	Compare	CMP and CMN, page 73
EOR	Exclusive OR	AND, ORR, EOR, BIC, and ORN, page 70
LSL	Logical Shift Left	ASR, LSL, LSR, ROR, and RRX, page 71
LSR	Logical Shift Right	ASR, LSL, LSR, ROR, and RRX, page 71
MOV	Move	MOV and MVN, page 73
MOVT	Move Top	MOVT, page 75
MOVW	Move 16-bit constant	MOV and MVN, page 73
MVN	Move NOT	"MOV and MVN" on page 73
ORN	Logical OR NOT	AND, ORR, EOR, BIC, and ORN, page 70
ORR	Logical OR	AND, ORR, EOR, BIC, and ORN, page 70
RBIT	Reverse Bits	REV, REV16, REVSH, and RBIT, page 75
REV	Reverse byte order in a word	REV, REV16, REVSH, and RBIT, page 75
REV16	Reverse byte order in each halfword	REV, REV16, REVSH, and RBIT, page 75
REVSH	Reverse byte order in bottom halfword and sign extend	REV, REV16, REVSH, and RBIT, page 75
ROR	Rotate Right	ASR, LSL, LSR, ROR, and RRX, page 71
RRX	Rotate Right with Extend	ASR, LSL, LSR, ROR, and RRX, page 71
RSB	Reverse Subtract	ADD, ADC, SUB, SBC, and RSB, page 69
SBC	Subtract with Carry	ADD, ADC, SUB, SBC, and RSB, page 69
SUB	Subtract	ADD, ADC, SUB, SBC, and RSB, page 69
SUBW	Subtract	ADD, ADC, SUB, SBC, and RSB, page 69
TEQ	Test Equivalence	TST and TEQ, page 76
TST	Test	TST and TEQ, page 76

3.6.5.1 ADD, ADC, SUB, SBC, and RSB

Add, Add with carry, Subtract, Subtract with carry, and Reverse Subtract.

3.6.5.1.1 Syntax

`op{S}{cond} {Rd,} Rn, Operand2`

`op{cond} {Rd,} Rn, #imm12` ; ADD and SUB only

where:

- *op* is one of:
 - ADD: Add
 - ADC: Add with Carry
 - SUB: Subtract
 - SBC: Subtract with Carry
 - RSB: Reverse Subtract
- *S* is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see [Conditional Execution](#), page 55.
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *Rd* is the destination register. If *Rd* is omitted, the destination register is *Rn*.
- *Rn* is the register holding the first operand.
- *Operand2* is a flexible second operand. See [Flexible Second Operand](#), page 51 for details of the options.
- *imm12* is any value in the range 0-4095.

3.6.5.1.2 Operation

The ADD instruction adds the value of *Operand2* or *imm12* to the value in *Rn*.

The ADC instruction adds the values in *Rn* and *Operand2*, together with the carry flag.

The SUB instruction subtracts the value of *Operand2* or *imm12* from the value in *Rn*.

The SBC instruction subtracts the value of *Operand2* from the value in *Rn*. If the carry flag is clear, the result is reduced by one.

The RSB instruction subtracts the value in *Rn* from the value of *Operand2*. This is useful because of the wide range of options for *Operand2*.

Use ADC and SBC to synthesize multiword arithmetic, see [Multiword Arithmetic Examples](#), page 70.

See also [ADR](#), page 59.

Note: ADDW is equivalent to the ADD syntax that uses the *imm12* operand. SUBW is equivalent to the SUB syntax that uses the *imm12* operand.

3.6.5.1.3 Restrictions

In these instructions:

- *Operand2* must not be SP and must not be PC
- *Rd* can be SP only in ADD and SUB, and only with the additional restrictions:
 - *Rn* must also be SP
 - any shift in *Operand2* must be limited to a maximum of 3 bits using LSL
- *Rn* can be SP only in ADD and SUB
- *Rd* can be PC only in the ADD{*cond*} PC, PC, Rm instruction where:
 - you must not specify the *S* suffix
 - *Rm* must not be PC and must not be SP
 - if the instruction is conditional, it must be the last instruction in the IT block
- with the exception of the ADD{*cond*} PC, PC, Rm instruction, *Rn* can be PC only in ADD and SUB, and only with the additional restrictions:
 - you must not specify the *S* suffix
 - the second operand must be a constant in the range 0 to 4095.
 - When using the PC for an addition or a subtraction, bits[1:0] of the PC are rounded to b00 before performing the calculation, making the base address for the calculation word-aligned.

- If you want to generate the address of an instruction, you have to adjust the constant based on the value of the PC. ARM recommends that you use the ADR instruction instead of ADD or SUB with *Rn* equal to the PC, because your assembler automatically calculates the correct constant for the ADR instruction.

When *Rd* is PC in the ADD{*cond*} PC, PC, *Rm* instruction:

- bit[0] of the value written to the PC is ignored
- a branch occurs to the address created by forcing bit[0] of that value to 0.

3.6.5.1.4 Condition Flags

If S is specified, these instructions update the N, Z, C and V flags according to the result.

Examples

```
ADD    R2, R1, R3
SUBS   R8, R6, #240      ; Sets the flags on the result
RSB    R4, R4, #1280     ; Subtracts contents of R4 from 1280
ADCHI  R11, R0, R3       ; Only executed if C flag set and Z
                        ; flag clear.
```

3.6.5.1.5 Multiword Arithmetic Examples

The following example shows two instructions that add a 64-bit integer contained in R2 and R3 to another 64-bit integer contained in R0 and R1, and place the result in R4 and R5.

Example 4

64-bit addition

```
ADDS   R4, R0, R2      ; add the least significant words
ADC    R5, R1, R3      ; add the most significant words with carry
```

Multiword values do not have to use consecutive registers. The following example shows instructions that subtract a 96-bit integer contained in R9, R1, and R11 from another contained in R6, R2, and R8. The example stores the result in R6, R9, and R2.

Example 5

96-bit subtraction

```
SUBS   R6, R6, R9      ; subtract the least significant words
SBCS   R9, R2, R1      ; subtract the middle words with carry
SBC    R2, R8, R11     ; subtract the most significant words with carry
```

3.6.5.2 AND, ORR, EOR, BIC, and ORN

Logical AND, OR, Exclusive OR, Bit Clear, and OR NOT.

3.6.5.2.1 Syntax

op{S}{*cond*} {*Rd*,} *Rn*, Operand2

where:

- *op* is one of:
 - AND: logical AND.
 - ORR: logical OR, or bit set.
 - EOR: logical Exclusive OR.
 - BIC: logical AND NOT, or bit clear.
 - ORN: logical OR NOT.
- *S* is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see [Conditional Execution](#), page 55.
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.

- *Rd* is the destination register.
- *Rn* is the register holding the first operand.
- *Operand2* is a flexible second operand. See [Flexible Second Operand](#), page 51 for details of the options.

3.6.5.2.2 Operation

The AND, EOR, and ORR instructions perform bitwise AND, Exclusive OR, and OR operations on the values in *Rn* and *Operand2*.

The BIC instruction performs an AND operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

The ORN instruction performs an OR operation on the bits in *Rn* with the complements of the corresponding bits in the value of *Operand2*.

3.6.5.2.3 Restrictions

Do not use SP and do not use PC.

3.6.5.2.4 Condition Flags

If S is specified, these instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see [Flexible Second Operand](#), page 51
- do not affect the V flag.

Examples

```

AND      R9, R2, #0xFF00
ORREQ    R2, R0, R5
ANDS     R9, R8, #0x19
EORS     R7, R11, #0x18181818
BIC      R0, R1, #0xab
ORN      R7, R11, R14, ROR #4
ORNS     R7, R11, R14, ASR #32

```

3.6.5.3 ASR, LSL, LSR, ROR, and RRX

Arithmetic Shift Right, Logical Shift Left, Logical Shift Right, Rotate Right, and Rotate Right with Extend.

3.6.5.3.1 Syntax

```

op{S}{cond} Rd, Rm, Rs
op{S}{cond} Rd, Rm, #n
RRX{S}{cond} Rd, Rm

```

where:

- op is one of:
 - ASR: Arithmetic Shift Right.
 - LSL: Logical Shift Left.
 - LSR: Logical Shift Right.
 - ROR: Rotate Right.
- S is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation, see [Conditional Execution](#), page 55.
- *Rd* is the destination register.
- *Rm* is the register holding the value to be shifted.
- *Rs* is the register holding the shift length to apply to the value in *Rm*. Only the least significant byte is used and can be in the range 0 to 255.

- n is the shift length. The range of shift length depends on the instruction:
 - ASR shift length from 1 to 32
 - LSL shift length from 0 to 31
 - LSR shift length from 1 to 32
 - ROR shift length from 1 to 31

Note: MOVs Rd, Rm is the preferred syntax for LSLS $Rd, Rm, \#0$.

3.6.5.3.2 Operation

ASR, LSL, LSR, and ROR move the bits in the register Rm to the left or right by the number of places specified by constant n or register Rs .

RRX moves the bits in register Rm to the right by 1.

In all these instructions, the result is written to Rd , but the value in register Rm remains unchanged. For details on what result is generated by the different instructions, see [Shift Operations](#), page 52.

3.6.5.3.3 Restrictions

Do not use SP and do not use PC.

3.6.5.3.4 Condition Flags

If S is specified:

- these instructions update the N and Z flags according to the result
- the C flag is updated to the last bit shifted out, except when the shift length is 0, see [Shift Operations](#), page 52.

Examples

```
ASR    R7, R8, #9 ; Arithmetic shift right by 9 bits
LSLS   R1, R2, #3 ; Logical shift left by 3 bits with flag update
LSR    R4, R5, #6 ; Logical shift right by 6 bits
ROR    R4, R5, R6 ; Rotate right by the value in the bottom byte of R6
RRX    R4, R5      ; Rotate right with extend.
```

3.6.5.4 CLZ

Count Leading Zeros.

3.6.5.4.1 Syntax

CLZ{cond} Rd, Rm

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register.

Rm is the operand register.

3.6.5.4.2 Operation

The CLZ instruction counts the number of leading zeros in the value in Rm and returns the result in Rd . The result value is 32 if no bits are set and zero if bit[31] is set.

3.6.5.4.3 Restrictions

Do not use SP, and do not use PC.

3.6.5.4.4 Condition Flags

This instruction does not change the flags.

Examples

```
CLZ      R4, R9
```

```
CLZNE    R2, R3
```

3.6.5.5 CMP and CMN

Compare and Compare Negative.

3.6.5.5.1 Syntax

```
CMP{cond} Rn, Operand2
```

```
CMN{cond} Rn, Operand2
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See [Flexible Second Operand](#), page 51 for details of the options.

3.6.5.5.2 Operation

These instructions compare the value in a register with *Operand2*. They update the condition flags on the result, but do not write the result to a register.

The CMP instruction subtracts the value of *Operand2* from the value in *Rn*. This is the same as a SUBS instruction, except that the result is discarded.

The CMN instruction adds the value of *Operand2* to the value in *Rn*. This is the same as an ADDS instruction, except that the result is discarded.

3.6.5.5.3 Restrictions

In these instructions:

- do not use PC
- *Operand2* must not be SP.

3.6.5.5.4 Condition Flags

These instructions update the N, Z, C and V flags according to the result.

Examples

```
CMP      R2, R9
```

```
CMN      R0, #6400
```

```
CMPGT    SP, R7, LSL #2
```

3.6.5.6 MOV and MVN

Move and Move NOT.

3.6.5.6.1 Syntax

```
MOV{S}{cond} Rd, Operand2
```

```
MOV{cond} Rd, #imm16
```

```
MVN{S}{cond} Rd, Operand2
```

where:

S is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see [Conditional Execution](#), page 55.

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register.

Operand2 is a flexible second operand. See [Flexible Second Operand](#), page 51 for details of the options.

imm16 is any value in the range 0-65535.

3.6.5.6.2 Operation

The MOV instruction copies the value of *Operand2* into *Rd*.

When *Operand2* in a MOV instruction is a register with a shift other than LSL #0, the preferred syntax is the corresponding shift instruction:

- ASR{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, ASR #n
- LSL{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, LSL #n if *n* != 0
- LSR{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, LSR #n
- ROR{S}{cond} Rd, Rm, #n is the preferred syntax for MOV{S}{cond} Rd, Rm, ROR #n
- RRX{S}{cond} Rd, Rm is the preferred syntax for MOV{S}{cond} Rd, Rm, RRX.

Also, the MOV instruction permits additional forms of *Operand2* as synonyms for shift instructions:

- MOV{S}{cond} Rd, Rm, ASR Rs is a synonym for ASR{S}{cond} Rd, Rm, Rs
- MOV{S}{cond} Rd, Rm, LSL Rs is a synonym for LSL{S}{cond} Rd, Rm, Rs
- MOV{S}{cond} Rd, Rm, LSR Rs is a synonym for LSR{S}{cond} Rd, Rm, Rs
- MOV{S}{cond} Rd, Rm, ROR Rs is a synonym for ROR{S}{cond} Rd, Rm, Rs

See [ASR, LSL, LSR, ROR, and RRX](#), page 71.

The MVN instruction takes the value of *Operand2*, performs a bitwise logical NOT operation on the value, and places the result into *Rd*.

Note: The MOVW instruction provides the same function as MOV, but is restricted to using the *imm16* operand.

3.6.5.6.3 Restrictions

You can use SP and PC only in the MOV instruction, with the following restrictions:

- the second operand must be a register without shift
- you must not specify the S suffix.

When *Rd* is PC in a MOV instruction:

- bit[0] of the value written to the PC is ignored
- a branch occurs to the address created by forcing bit[0] of that value to 0.

Note: Though it is possible to use MOV as a branch instruction, ARM strongly recommends the use of a BX or BLX instruction to branch for software portability to the ARM instruction set.

3.6.5.6.4 Condition Flags

- If S is specified, these instructions:
- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see [Flexible Second Operand](#), page 51
- do not affect the V flag.

Example

```
MOVS R11, #0x000B    ; Write value of 0x000B to R11, flags get updated
MOV  R1, #0xFA05     ; Write value of 0xFA05 to R1, flags are not updated
MOVS R10, R12        ; Write value in R12 to R10, flags get updated
MOV  R3, #23         ; Write value of 23 to R3
MOV  R8, SP          ; Write value of stack pointer to R8
MVNS R2, #0xF        ; Write value of 0xFFFFFFFF0 (bitwise inverse of 0xF)
                        ; to the R2 and update flags.
```

3.6.5.7 MOV

Move Top.

3.6.5.7.1 Syntax

`MOVT{cond} Rd, #imm16`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register.

imm16 is a 16-bit immediate constant.

3.6.5.7.2 Operation

MOVT writes a 16-bit immediate value, *imm16*, to the top halfword, *Rd*[31:16], of its destination register. The write does not affect *Rd*[15:0].

The MOV, MOVT instruction pair enables you to generate any 32-bit constant.

3.6.5.7.3 Restrictions

Rd must not be SP and must not be PC.

3.6.5.7.4 Condition Flags

This instruction does not change the flags.

Examples

```
MOVT    R3, #0xF123 ; Write 0xF123 to upper halfword of R3, lower halfword
                    ; and APSR are unchanged.
```

3.6.5.8 REV, REV16, REVSH, and RBIT

Reverse bytes and Reverse bits.

3.6.5.8.1 Syntax

`op{cond} Rd, Rn`

where:

- *op* is any of:
 - REV: Reverse byte order in a word.
 - REV16: Reverse byte order in each halfword independently.
 - REVSH: Reverse byte order in the bottom halfword, and sign extend to 32 bits.
 - RBIT: Reverse the bit order in a 32-bit word.
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *Rd* is the destination register.
- *Rn* is the register holding the operand.

3.6.5.8.2 Operation

Use these instructions to change endianness of data:

- REV: converts 32-bit big-endian data into little-endian data or 32-bit little-endian data into big-endian data.
- REV16: converts 16-bit big-endian data into little-endian data or 16-bit little-endian data into big-endian data.
- REVSH: converts either:
 - 16-bit signed big-endian data into 32-bit signed little-endian data
 - 16-bit signed little-endian data into 32-bit signed big-endian data.

3.6.5.8.3 Restrictions

Do not use SP and do not use PC.

3.6.5.8.4 Condition Flags

These instructions do not change the flags.

Examples

```
REV    R3, R7 ; Reverse byte order of value in R7 and write it to R3
REV16  R0, R0 ; Reverse byte order of each 16-bit halfword in R0
REVSH  R0, R5 ; Reverse Signed Halfword
REVHS  R3, R7 ; Reverse with Higher or Same condition
RBIT   R7, R8 ; Reverse bit order of value in R8 and write the result to R7.
```

3.6.5.9 TST and TEQ

Test bits and Test Equivalence.

3.6.5.9.1 Syntax

```
TST{cond} Rn, Operand2
```

```
TEQ{cond} Rn, Operand2
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rn is the register holding the first operand.

Operand2 is a flexible second operand. See [Flexible Second Operand](#), page 51 for details of the options.

3.6.5.9.2 Operation

These instructions test the value in a register against *Operand2*. They update the condition flags based on the result, but do not write the result to a register.

The TST instruction performs a bitwise AND operation on the value in *Rn* and the value of *Operand2*. This is the same as the ANDS instruction, except that it discards the result.

To test whether a bit of *Rn* is 0 or 1, use the TST instruction with an *Operand2* constant that has that bit set to 1 and all other bits cleared to 0.

The TEQ instruction performs a bitwise Exclusive OR operation on the value in *Rn* and the value of *Operand2*. This is the same as the EORS instruction, except that it discards the result.

Use the TEQ instruction to test if two values are equal without affecting the V or C flags.

TEQ is also useful for testing the sign of a value. After the comparison, the N flag is the logical Exclusive OR of the sign bits of the two operands.

3.6.5.9.3 Restrictions

Do not use SP and do not use PC.

3.6.5.9.4 Condition Flags

These instructions:

- update the N and Z flags according to the result
- can update the C flag during the calculation of *Operand2*, see [Flexible Second Operand](#), page 51
- do not affect the V flag.

Examples

```
TST    R0, #0x3F8 ; Perform bitwise AND of R0 value to 0x3F8,
                  ; APSR is updated but result is discarded
TEQEQ  R10, R9    ; Conditionally test if value in R10 is equal to
                  ; value in R9, APSR is updated but result is discarded.
```

3.6.6 Multiply and Divide Instructions

The following table shows the multiply and divide instructions:

Table 34 • Multiply and Divide Instructions

Mnemonic	Brief Description	See
MLA	Multiply with Accumulate, 32-bit result	"MUL, MLA, and MLS" on page 71
MLS	Multiply and Subtract, 32-bit result	"MUL, MLA, and MLS" on page 71
MUL	Multiply, 32-bit result	"MUL, MLA, and MLS" on page 71
SDIV	Signed Divide	"SDIV and UDIV" on page 73
SMLAL	Signed Multiply with Accumulate (32x32+64), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 72
SMULL	Signed Multiply (32x32), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 72
UDIV	Unsigned Divide	"SDIV and UDIV" on page 73
UMLAL	Unsigned Multiply with Accumulate (32x32+64), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 72
UMULL	Unsigned Multiply (32x32), 64-bit result	"UMULL, UMLAL, SMULL, and SMLAL" on page 72

3.6.6.1 MUL, MLA, and MLS

Multiply, Multiply with Accumulate, and Multiply with Subtract, using 32-bit operands, and producing a 32-bit result.

3.6.6.1.1 Syntax

`MUL{S}{cond} {Rd}, {Rn}, {Rm} ; Multiply`

`MLA{cond} {Rd}, {Rn}, {Rm}, {Ra} ; Multiply with accumulate`

`MLS{cond} {Rd}, {Rn}, {Rm}, {Ra} ; Multiply with subtract`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

S is an optional suffix. If *S* is specified, the condition code flags are updated on the result of the operation, see [Conditional Execution](#), page 55.

Rd is the destination register. If *Rd* is omitted, the destination register is *Rn*.

Rn, *Rm* are registers holding the values to be multiplied.

Ra is a register holding the value to be added or subtracted from.

3.6.6.1.2 Operation

The MUL instruction multiplies the values from *Rn* and *Rm*, and places the least significant 32 bits of the result in *Rd*.

The MLA instruction multiplies the values from *Rn* and *Rm*, adds the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The MLS instruction multiplies the values from *Rn* and *Rm*, subtracts the product from the value from *Ra*, and places the least significant 32 bits of the result in *Rd*.

The results of these instructions do not depend on whether the operands are signed or unsigned.

3.6.6.1.3 Restrictions

In these instructions, do not use SP and do not use PC.

If you use the S suffix with the MUL instruction:

- *Rd*, *Rn*, and *Rm* must all be in the range R0 to R7
- *Rd* must be the same as *Rm*
- you must not use the *cond* suffix.

3.6.6.1.4 Condition Flags

If S is specified, the MUL instruction:

- updates the N and Z flags according to the result
- does not affect the C and V flags.

Examples

```
MUL    R10, R2, R5      ; Multiply, R10 = R2 x R5
MLA    R10, R2, R1, R5  ; Multiply with accumulate, R10 = (R2 x R1) + R5
MULS   R0, R2, R2       ; Multiply with flag update, R0 = R2 x R2
MULLT  R2, R3, R2       ; Conditionally multiply, R2 = R3 x R2
MLS    R4, R5, R6, R7   ; Multiply with subtract, R4 = R7 - (R5 x R6).
```

3.6.6.2 UMULL, UMLAL, SMULL, and SMLAL

Signed and Unsigned Long Multiply, with optional Accumulate, using 32-bit operands and producing a 64-bit result.

3.6.6.2.1 Syntax

```
op{cond} RdLo, RdHi, Rn, Rm
```

where:

- *op* is one of:
 - UMULL: Unsigned Long Multiply
 - UMLAL: Unsigned Long Multiply, with Accumulate
 - SMULL: Signed Long Multiply
 - SMLAL: Signed Long Multiply, with Accumulate
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *RdHi*, *RdLo* are the destination registers. For UMLAL and SMLAL they also hold the accumulating value.
- *Rn*, *Rm* are registers holding the operands.

3.6.6.2.2 Operation

The UMULL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The UMLAL instruction interprets the values from *Rn* and *Rm* as unsigned integers. It multiplies these integers, adds the 64-bit result to the 64-bit unsigned integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

The SMULL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers and places the least significant 32 bits of the result in *RdLo*, and the most significant 32 bits of the result in *RdHi*.

The SMLAL instruction interprets the values from *Rn* and *Rm* as two's complement signed integers. It multiplies these integers, adds the 64-bit result to the 64-bit signed integer contained in *RdHi* and *RdLo*, and writes the result back to *RdHi* and *RdLo*.

3.6.6.2.3 Restrictions

In these instructions:

- do not use SP and do not use PC
- *RdHi* and *RdLo* must be different registers.

3.6.6.2.4 Condition Flags

These instructions do not affect the condition code flags.

Examples

```
UMULL      R0, R4, R5, R6    ; Unsigned (R4,R0) = R5 x R6
```

```
SMLAL      R4, R5, R3, R8    ; Signed (R5,R4) = (R5,R4) + R3 x R8
```

3.6.6.3 SDIV and UDIV

Signed Divide and Unsigned Divide.

3.6.6.3.1 Syntax

```
SDIV{cond} {Rd,} Rn, Rm
```

```
UDIV{cond} {Rd,} Rn, Rm
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register. If *Rd* is omitted, the destination register is *Rn*.

Rn is the register holding the value to be divided.

Rm is a register holding the divisor.

3.6.6.3.2 Operation

SDIV performs a signed integer division of the value in *Rn* by the value in *Rm*.

UDIV performs an unsigned integer division of the value in *Rn* by the value in *Rm*.

For both instructions, if the value in *Rn* is not divisible by the value in *Rm*, the result is rounded towards zero.

3.6.6.3.3 Restrictions

Do not use SP and do not use PC.

3.6.6.3.4 Condition Flags

These instructions do not change the flags.

Examples

```
SDIV R0, R2, R4    ; Signed divide, R0 = R2/R4
```

```
UDIV R8, R8, R1    ; Unsigned divide, R8 = R8/R1.
```

3.6.7 Saturating Instructions

This section describes the saturating instructions, SSAT and USAT.

3.6.7.1 SSAT and USAT

Signed Saturate and Unsigned Saturate to any bit position, with optional shift before saturating.

3.6.7.1.1 Syntax

`op{cond} Rd, #n, Rm {, shift #s}`

where:

- *op* is one of:
 - SSAT: Saturates a signed value to a signed range.
 - USAT: Saturates a signed value to an unsigned range.
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *Rd* is the destination register.
- *n* specifies the bit position to saturate to:
 - *n* ranges from 1 to 32 for SSAT
 - *n* ranges from 0 to 31 for USAT.
- *Rm* is the register containing the value to saturate.
- *shift #s* is an optional shift applied to *Rm* before saturating. It must be one of the following:
 - ASR #s where *s* is in the range 1 to 31
 - LSL #s where *s* is in the range 0 to 31

3.6.7.1.2 Operation

These instructions saturate to a signed or unsigned *n*-bit value.

The SSAT instruction applies the specified shift, then saturates to the signed range $-2^{n-1} \leq x \leq 2^{n-1}-1$.

The USAT instruction applies the specified shift, then saturates to the unsigned range $0 \leq x \leq 2^n-1$.

For signed *n*-bit saturation using SSAT, this means that:

- if the value to be saturated is less than -2^{n-1} , the result returned is -2^{n-1}
- if the value to be saturated is greater than $2^{n-1}-1$, the result returned is $2^{n-1}-1$
- otherwise, the result returned is the same as the value to be saturated.

For unsigned *n*-bit saturation using USAT, this means that:

- if the value to be saturated is less than 0, the result returned is 0
- if the value to be saturated is greater than 2^n-1 , the result returned is 2^n-1
- otherwise, the result returned is the same as the value to be saturated.

If the returned result is different from the value to be saturated, it is called *saturation*. If saturation occurs, the instruction sets the Q flag to 1 in the APSR. Otherwise, it leaves the Q flag unchanged. To clear the Q flag to 0, you must use the MSR instruction, see [MSR](#), page 92.

To read the state of the Q flag, use the MRS instruction, see [MRS](#), page 91.

3.6.7.1.3 Restrictions

Do not use SP and do not use PC.

3.6.7.1.4 Condition Flags

These instructions do not affect the condition code flags.

If saturation occurs, these instructions set the Q flag to 1.

Examples

```
SSAT    R7, #16, R7, LSL #4 ; Logical shift left value in R7 by 4, then
                                ; saturate it as a signed 16-bit value and
                                ; write it back to R7

USATNE  R0, #7, R5          ; Conditionally saturate value in R5 as an
                                ; unsigned 7 bit value and write it to R0.
```

3.6.8 Bitfield instructions

The following table shows the instructions that operate on adjacent sets of bits in registers or bitfields:

Table 35 • Packing and Unpacking Instructions

Mnemonic	Brief description	See
BFC	Bit Field Clear	BFC and BFI , page 81
BFI	Bit Field Insert	BFC and BFI , page 81
SBFX	Signed Bit Field Extract	SBFX and UBFX , page 81
SXTB	Sign extend a byte	SXT and UXT , page 82
SXTH	Sign extend a halfword	SXT and UXT , page 82
UBFX	Unsigned Bit Field Extract	SBFX and UBFX , page 81
UXTB	Zero extend a byte	SXT and UXT , page 82
UXTH	Zero extend a halfword	SXT and UXT , page 82

3.6.8.1 BFC and BFI

Bit Field Clear and Bit Field Insert.

3.6.8.1.1 Syntax

`BFC{cond} Rd, #lsb, #width`

`BFI{cond} Rd, Rn, #lsb, #width`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register.

Rn is the source register.

lsb is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

width is the width of the bitfield and must be in the range 1 to 32-*lsb*.

3.6.8.1.2 Operation

BFC clears a bitfield in a register. It clears *width* bits in *Rd*, starting at the low bit position *lsb*. Other bits in *Rd* are unchanged.

BFI copies a bitfield into one register from another register. It replaces *width* bits in *Rd* starting at the low bit position *lsb*, with *width* bits from *Rn* starting at bit[0]. Other bits in *Rd* are unchanged.

3.6.8.1.3 Restrictions

Do not use SP and do not use PC.

3.6.8.1.4 Condition Flags

These instructions do not affect the flags.

Examples

`BFC R4, #8, #12 ; Clear bit 8 to bit 19 (12 bits) of R4 to 0`

`BFI R9, R2, #8, #12 ; Replace bit 8 to bit 19 (12 bits) of R9 with
; bit 0 to bit 11 from R2.`

3.6.8.2 SBFX and UBFX

Signed Bit Field Extract and Unsigned Bit Field Extract.

3.6.8.2.1 Syntax

SBFX{cond} Rd, Rn, #lsb, #width

UBFX{cond} Rd, Rn, #lsb, #width

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register.

Rn is the source register.

lsb is the position of the least significant bit of the bitfield. *lsb* must be in the range 0 to 31.

width is the width of the bitfield and must be in the range 1 to 32-*lsb*.

3.6.8.2.2 Operation

SBFX extracts a bitfield from one register, sign extends it to 32 bits, and writes the result to the destination register.

UBFX extracts a bitfield from one register, zero extends it to 32 bits, and writes the result to the destination register.

3.6.8.2.3 Restrictions

Do not use SP and do not use PC.

3.6.8.2.4 Condition Flags

These instructions do not affect the flags.

Examples

SBFX R0, R1, #20, #4 ; Extract bit 20 to bit 23 (4 bits) from R1 and sign
; extend to 32 bits and then write the result to R0.

UBFX R8, R11, #9, #10 ; Extract bit 9 to bit 18 (10 bits) from R11 and zero
; extend to 32 bits and then write the result to R8.

3.6.8.3 SXT and UXT

Sign extend and Zero extend.

3.6.8.3.1 Syntax

SXTextend{cond} {Rd,} Rm {, ROR #n}

UXTextend{cond} {Rd,} Rm {, ROR #n}

where:

- extend is one of:
 - B: Extends an 8-bit value to a 32-bit value.
 - H: Extends a 16-bit value to a 32-bit value.
- *cond* is an optional condition code, see [Conditional Execution](#), page 55.
- *Rd* is the destination register.
- *Rm* is the register holding the value to extend.
- ROR #*n* is one of:
 - ROR #8: Value from *Rm* is rotated right 8 bits.
 - ROR #16: Value from *Rm* is rotated right 16 bits.
 - ROR #24: Value from *Rm* is rotated right 24 bits.
 - If ROR #*n* is omitted, no rotation is performed.

3.6.8.3.2 Operation

These instructions do the following:

1. Rotate the value from *Rm* right by 0, 8, 16 or 24 bits.
2. Extract bits from the resulting value:
 - SXTB extracts bits [7:0] and sign extends to 32 bits.
 - UXTB extracts bits [7:0] and zero extends to 32 bits.
 - SXTB extracts bits [15:0] and sign extends to 32 bits.
 - UXTB extracts bits [15:0] and zero extends to 32 bits.

3.6.8.3.3 Restrictions

Do not use SP and do not use PC.

3.6.8.3.4 Condition Flags

These instructions do not affect the flags.

Examples

```
SXTB  R4, R6, ROR #16 ; Rotate R6 right by 16 bits, then obtain the lower
                        ; halfword of the result and then sign extend to
                        ; 32 bits and write the result to R4.
```

```
UXTB  R3, R10          ; Extract lowest byte of the value in R10 and zero
                        ; extend it, and write the result to R3.
```

3.6.9 Branch and Control Instructions

The following table lists the branch and control instructions:

Table 36 • Branch and Control Instructions

Mnemonic	Brief description	See
B	Branch	B, BL, BX, and BLX , page 83
BL	Branch with Link	B, BL, BX, and BLX , page 83
BLX	Branch indirect with Link	B, BL, BX, and BLX , page 83
BX	Branch indirect	B, BL, BX, and BLX , page 83
CBNZ	Compare and Branch if Non Zero	CBZ and CBNZ , page 85
CBZ	Compare and Branch if Zero	CBZ and CBNZ , page 85
IT	If-Then	IT , page 85
TBB	Table Branch Byte	TBB and TBH , page 87
TBH	Table Branch Halfword	TBB and TBH , page 87

3.6.9.1 B, BL, BX, and BLX

Branch instructions.

3.6.9.1.1 Syntax

```
B{cond} label
```

```
BL{cond} label
```

```
BX{cond} Rm
```

```
BLX{cond} Rm
```

where:

B is branch (immediate).

BL is branch with link (immediate).

BX is branch indirect (register).

BLX is branch indirect with link (register).

cond is an optional condition code, see [Conditional Execution](#), page 55.

label is a PC-relative expression. See [PC-relative Expressions](#), page 55.

Rm is a register that indicates an address to branch to. Bit[0] of the value in *Rm* must be 1, but the address to branch to is created by changing bit[0] to 0.

3.6.9.1.2 Operation

All these instructions cause a branch to *label*, or to the address indicated in *Rm*. In addition:

- The BL and BLX instructions write the address of the next instruction to LR (the link register, R14).
- The BX and BLX instructions result in a UsageFault exception if bit[0] of *Rm* is 0.

Bcond label is the only conditional instruction that can be either inside or outside an IT block. All other branch instructions can only be conditional inside an IT block, and are always unconditional otherwise, see [IT](#), page 85.

The following table lists the ranges for the various branch instructions.

Table 37 • Branch Ranges

Instruction	Branch Range
B label	-16 MB to +16 MB
Bcond label (outside IT block)	-1 MB to +1 MB
Bcond label (inside IT block)	-16 MB to +16 MB
BL{cond} label	-16 MB to +16 MB
BX{cond} Rm	Any value in register
BLX{cond} Rm	Any value in register

You may have to use the .W suffix to get the maximum branch range. See [Instruction Width Selection](#), page 58.

3.6.9.1.3 Restrictions

The restrictions are:

- do not use PC in the BLX instruction
- for BX and BLX, bit[0] of *Rm* must be 1 for correct execution but a branch occurs to the target address created by changing bit[0] to 0
- when any of these instructions is inside an IT block, it must be the last instruction of the IT block.

Note: *Bcond* is the only conditional instruction that is not required to be inside an IT block. However, it has a longer branch range when it is inside an IT block.

3.6.9.1.4 Condition Flags

These instructions do not change the flags.

Examples

```

B      loopA ; Branch to loopA
BLE    ng    ; Conditionally branch to label ng
B.W    target ; Branch to target within 16MB range
BEQ    target ; Conditionally branch to target
BEQ.W  target ; Conditionally branch to target within 1MB
```

```

BL      funC    ; Branch with link (Call) to function funC, return address
                ; stored in LR
BX      LR      ; Return from function call
BXNE    R0      ; Conditionally branch to address stored in R0
BLX     R0      ; Branch with link and exchange (Call) to a address stored
                ; in R0.

```

3.6.9.2 CBZ and CBNZ

Compare and Branch on Zero, Compare and Branch on Non-Zero.

3.6.9.2.1 Syntax

```
CBZ Rn, label
```

```
CBNZ Rn, label
```

where:

Rn is the register holding the operand.

label is the branch destination.

3.6.9.2.2 Operation

Use the CBZ or CBNZ instructions to avoid changing the condition code flags and to reduce the number of instructions.

CBZ *Rn*, *label* does not change condition flags but is otherwise equivalent to:

```
CMP Rn, #0
```

```
BEQ label
```

CBNZ *Rn*, *label* does not change condition flags but is otherwise equivalent to:

```
CMP Rn, #0
```

```
BNE label
```

3.6.9.2.3 Restrictions

The restrictions are:

- *Rn* must be in the range of R0 to R7
- the branch destination must be within 4 to 130 bytes after the instruction
- these instructions must not be used inside an IT block.

3.6.9.2.4 Condition Flags

These instructions do not change the flags.

Examples

```
CBZ R5, target ; Forward branch if R5 is zero
```

```
CBNZ R0, target ; Forward branch if R0 is not zero.
```

3.6.9.3 IT

If-Then condition instruction.

3.6.9.3.1 Syntax

```
IT{x{y{z}}} cond
```

where:

x specifies the condition switch for the second instruction in the IT block.

y specifies the condition switch for the third instruction in the IT block.

z specifies the condition switch for the fourth instruction in the IT block.

cond specifies the condition for the first instruction in the IT block. The condition switch for the second, third and fourth instruction in the IT block can be either:

- T: Then. Applies the condition *cond* to the instruction.
- E: Else. Applies the inverse condition of *cond* to the instruction.

Note: It is possible to use AL (the *always* condition) for *cond* in an IT instruction. If this is done, all of the instructions in the IT block must be unconditional, and each of *x*, *y*, and *z* must be T or omitted but not E.

3.6.9.3.2 Operation

The IT instruction makes up to four following instructions conditional. The conditions can be all the same, or some of them can be the logical inverse of the others. The conditional instructions following the IT instruction form the *IT block*.

The instructions in the IT block, including any branches, must specify the condition in the {*cond*} part of their syntax.

Note: Your assembler might be able to generate the required IT instructions for conditional instructions automatically, so that you do not need to write them yourself. See your assembler documentation for details.

A BKPT instruction in an IT block is always executed, even if its condition fails.

Exceptions can be taken between an IT instruction and the corresponding IT block, or within an IT block. Such an exception results in entry to the appropriate exception handler, with suitable return information in LR and stacked PSR.

Instructions designed for use for exception returns can be used as normal to return from the exception, and execution of the IT block resumes correctly. This is the only way that a PC-modifying instruction is permitted to branch to an instruction in an IT block.

3.6.9.3.3 Restrictions

The following instructions are not permitted in an IT block:

- IT
- CBZ and CBNZ
- CPSID and CPSIE
- MOVS.N Rd, Rm.

Other restrictions when using an IT block are:

- a branch or any instruction that modifies the PC must either be outside an IT block or must be the last instruction inside the IT block. These are:
 - ADD PC, PC, Rm
 - MOV PC, Rm
 - B, BL, BX, BLX
 - any LDM, LDR, or POP instruction that writes to the PC
 - TBB and TBH
- do not branch to any instruction inside an IT block, except when returning from an exception handler
- all conditional instructions except *Bcond* must be inside an IT block. *Bcond* can be either outside or inside an IT block but has a larger branch range if it is inside one
- each instruction inside the IT block must specify a condition code suffix that is either the same or logical inverse as for the other instructions in the block.

Note: Your assembler might place extra restrictions on the use of IT blocks, such as prohibiting the use of assembler directives within them.

3.6.9.3.4 Condition Flags

This instruction does not change the flags.

Example

```

ITTE    NE                ; Next 3 instructions are conditional
ANDNE   R0, R0, R1        ; ANDNE does not update condition flags
ADDSNE  R2, R2, #1        ; ADDSNE updates condition flags
MOVEQ   R2, R3            ; Conditional move
CMP     R0, #9            ; Convert R0 hex value (0 to 15) into ASCII
                        ; ('0'-'9', 'A'-'F')

ITE     GT                ; Next 2 instructions are conditional
ADDGT   R1, R0, #55       ; Convert 0xA -> 'A'
ADDLE   R1, R0, #48       ; Convert 0x0 -> '0'

IT      GT                ; IT block with only one conditional instruction
ADDGT   R1, R1, #1        ; Increment R1 conditionally

ITTEE   EQ                ; Next 4 instructions are conditional
MOVEQ   R0, R1            ; Conditional move
ADDEQ   R2, R2, #10       ; Conditional add
ANDNE   R3, R3, #1        ; Conditional AND
BNE.W   dloop            ; Branch instruction can only be used in the last
                        ; instruction of an IT block

IT      NE                ; Next instruction is conditional
ADD     R0, R0, R1        ; Syntax error: no condition code used in IT block.

```

3.6.9.4 TBB and TBH

Table Branch Byte and Table Branch Halfword.

3.6.9.4.1 Syntax

```

TBB [Rn, Rm]
TBH [Rn, Rm, LSL #1]

```

where:

Rn is the register containing the address of the table of branch lengths. If *Rn* is PC, then the address of the table is the address of the byte immediately following the TBB or TBH instruction.

Rm is the index register. This contains an index into the table. For halfword tables, LSL #1 doubles the value in *Rm* to form the right offset into the table.

3.6.9.4.2 Operation

These instructions cause a PC-relative forward branch using a table of single byte offsets for TBB, or halfword offsets for TBH. *Rn* provides a pointer to the table, and *Rm* supplies an index into the table. For TBB the branch offset is twice the unsigned value of the byte returned from the table. and for TBH the branch offset is twice the unsigned value of the halfword returned from the table. The branch occurs to the address at that offset from the address of the byte immediately after the TBB or TBH instruction.

3.6.9.4.3 Restrictions

The restrictions are:

- *Rn* must not be SP
- *Rm* must not be SP and must not be PC
- when any of these instructions is used inside an IT block, it must be the last instruction of the IT block.

3.6.9.4.4 Condition Flags

These instructions do not change the flags.

Examples

```

ADR.W  R0, BranchTable_Byte

        TBB    [R0, R1]          ; R1 is the index, R0 is the base address of the
                                   ; branch table

Case1
; an instruction sequence follows
Case2
; an instruction sequence follows
Case3
; an instruction sequence follows

BranchTable_Byte
        DCB    0                  ; Case1 offset calculation
        DCB    ((Case2-Case1)/2) ; Case2 offset calculation
        DCB    ((Case3-Case1)/2) ; Case3 offset calculation

TBH     [PC, R1, LSL #1]         ; R1 is the index, PC is used as base of the
                                   ; branch table

BranchTable_H
        DCI     ((CaseA - BranchTable_H)/2) ; CaseA offset calculation
        DCI     ((CaseB - BranchTable_H)/2) ; CaseB offset calculation
        DCI     ((CaseC - BranchTable_H)/2) ; CaseC offset calculation

CaseA
; an instruction sequence follows
CaseB
; an instruction sequence follows
CaseC
; an instruction sequence follows

```

3.6.10 Miscellaneous Instructions

The following table lists the remaining Cortex-M3 processor instructions:

Table 38 • Miscellaneous Instructions

Mnemonic	Brief Description	See
BKPT	Breakpoint	BKPT , page 89
CPSID	Change Processor State, Disable Interrupts	CPS , page 89
CPSIE	Change Processor State, Enable Interrupts	CPS , page 89
DMB	Data Memory Barrier	DMB , page 90
DSB	Data Synchronization Barrier	DSB , page 91
ISB	Instruction Synchronization Barrier	ISB , page 91
MRS	Move from special register to register	MRS , page 91
MSR	Move from register to special register	MSR , page 92
NOP	No Operation	NOP , page 92
SEV	Send Event	SEV , page 93
SVC	Supervisor Call	SVC , page 93
WFE	Wait For Event	WFE , page 93
WFI	Wait For Interrupt	WFI , page 94

3.6.10.1 BKPT

Breakpoint.

3.6.10.1.1 Syntax

`BKPT #imm`

where:

imm is an expression evaluating to an integer in the range 0-255 (8-bit value).

3.6.10.1.2 Operation

The BKPT instruction causes the processor to enter Debug state. Debug tools can use this to investigate system state when the instruction at a particular address is reached.

imm is ignored by the processor. If required, a debugger can use it to store additional information about the breakpoint.

The BKPT instruction can be placed inside an IT block, but it executes unconditionally, unaffected by the condition specified by the IT instruction.

3.6.10.1.3 Condition Flags

This instruction does not change the flags.

Examples

```
BKPT #0x3    ; Breakpoint with immediate value set to 0x3 (debugger can
              ; extract the immediate value by locating it using the PC)
```

Note: ARM does not recommend the use of the BKPT instruction with an immediate value set to 0xAB for any purpose other than Semi-hosting.

3.6.10.2 CPS

Change Processor State.

3.6.10.2.1 Syntax

`CPSeffect iflags`

where:

- *effect* is one of:
 - IE: Clears the special purpose register.
 - ID: Sets the special purpose register.
- *iflags* is a sequence of one or more flags:
 - i: Set or clear PRIMASK.
 - f: Set or clear FAULTMASK.

3.6.10.2.2 Operation

CPS changes the PRIMASK and FAULTMASK special register values. See [Exception Mask Registers](#), page 25 for more information about these registers.

3.6.10.2.3 Restrictions

The restrictions are:

- use CPS only from privileged software, it has no effect if used in unprivileged software
- CPS cannot be conditional and so must not be used inside an IT block.

3.6.10.2.4 Condition Flags

This instruction does not change the condition flags.

Examples

```
CPSID i ; Disable interrupts and configurable fault handlers (set PRIMASK)
CPSID f ; Disable interrupts and all fault handlers (set FAULTMASK)
CPSIE i ; Enable interrupts and configurable fault handlers (clear PRIMASK)
CPSIE f ; Enable interrupts and fault handlers (clear FAULTMASK).
```

3.6.10.3 DMB

Data Memory Barrier.

3.6.10.3.1 Syntax

`DMB{cond}`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.10.3.2 Operation

DMB acts as a data memory barrier. It ensures that all explicit memory accesses that appear, in program order, before the DMB instruction are completed before any explicit memory accesses that appear, in program order, after the DMB instruction. DMB does not affect the ordering or execution of instructions that do not access memory.

3.6.10.3.3 Condition Flags

This instruction does not change the flags.

Examples

```
DMB ; Data Memory Barrier
```

3.6.10.4 DSB

Data Synchronization Barrier.

3.6.10.4.1 Syntax

`DSB{cond}`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.10.4.2 Operation

DSB acts as a special data synchronization memory barrier. Instructions that come after the DSB, in program order, do not execute until the DSB instruction completes. The DSB instruction completes when all explicit memory accesses before it complete.

3.6.10.4.3 Condition Flags

This instruction does not change the flags.

Examples

```
DSB ; Data Synchronisation Barrier
```

3.6.10.5 ISB

Instruction Synchronization Barrier.

3.6.10.5.1 Syntax

`ISB{cond}`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.10.5.2 Operation

ISB acts as an instruction synchronization barrier. It flushes the pipeline of the processor, so that all instructions following the ISB are fetched from cache or memory again, after the ISB instruction has been completed.

3.6.10.5.3 Condition Flags

This instruction does not change the flags.

Examples

```
ISB ; Instruction Synchronisation Barrier
```

3.6.10.6 MRS

Move the contents of a special register to a general-purpose register.

3.6.10.6.1 Syntax

`MRS{cond} Rd, spec_reg`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rd is the destination register.

spec_reg can be any of: APSR, IPSR, EPSR, IEPSPR, IAPSPR, EAPSPR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, or CONTROL.

Note: All the EPSR and IPSR fields are zero when read by the MRS instruction.

3.6.10.6.2 Operation

Use MRS in combination with MSR as part of a read-modify-write sequence for updating a PSR, for example to clear the Q flag.

Note: BASEPRI_MAX is an alias of BASEPRI when used with the MRS instruction. See [MSR](#), page 92.

3.6.10.6.3 Restrictions

Rd must not be SP and must not be PC.

3.6.10.6.4 Condition Flags

This instruction does not change the flags.

Examples

```
MRS R0, PRIMASK ; Read PRIMASK value and write it to R0.
```

3.6.10.7 MSR

Move the contents of a general-purpose register into the specified special register.

3.6.10.7.1 Syntax

```
MSR{cond} spec_reg, Rn
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

Rn is the source register.

spec_reg can be any of: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, or CONTROL.

Note: The processor ignores MSR writes to the EPSR and IPSR fields.

3.6.10.7.2 Operation

The register access operation in MSR depends on the privilege level. Unprivileged software can only access the APSR, see [Table 10](#), page 23. Privileged software can access all special registers.

In unprivileged software writes to unallocated or execution state bits in the PSR are ignored.

Note: When you write to BASEPRI_MAX, the instruction writes to BASEPRI only if either:

- *Rn* is non-zero and the current BASEPRI value is 0
- *Rn* is non-zero and less than the current BASEPRI value.

See [MRS](#), page 91

3.6.10.7.3 Restrictions

Rn must not be SP and must not be PC.

3.6.10.7.4 Condition Flags

This instruction updates the flags explicitly based on the value in *Rn*.

Examples

```
MSR CONTROL, R1 ; Read R1 value and write it to the CONTROL register.
```

3.6.10.8 NOP

No Operation.

3.6.10.8.1 Syntax

```
NOP{cond}
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.10.8.2 Operation

NOP does nothing. NOP is not necessarily a time-consuming NOP. The processor might remove it from the pipeline before it reaches the execution stage.

Use NOP for padding, for example to adjust the alignment of a following instruction.

3.6.10.8.3 Condition Flags

This instruction does not change the flags.

Examples

```
NOP ; No operation
```

3.6.10.9 SEV

Send Event.

3.6.10.9.1 Syntax

```
SEV{cond}
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.10.9.2 Operation

SEV is a hint instruction that causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register to 1, see [Power Management](#), page 45.

3.6.10.9.3 Condition Flags

This instruction does not change the flags.

Examples

```
SEV ; Send Event
```

3.6.10.10 SVC

Supervisor Call.

3.6.10.10.1 Syntax

```
SVC{cond} #imm
```

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

imm is an expression evaluating to an integer in the range 0-255 (8-bit value).

3.6.10.10.2 Operation

The SVC instruction causes the SVC exception.

imm is ignored by the processor. If required, it can be retrieved by the exception handler to determine what service is being requested.

3.6.10.10.3 Condition Flags

This instruction does not change the flags.

Examples

```
SVC #0x32 ; Supervisor Call (SVC handler can extract the immediate value
; by locating it via the stacked PC)
```

3.6.10.11 WFE

Wait For Event.

3.6.10.11.1 Syntax

`WFE{cond}`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.10.11.2 Operation

WFE is a hint instruction.

If the event register is 0, WFE suspends execution until one of the following events occurs:

- an exception, unless masked by the exception mask registers or the current priority level
- an exception enters the Pending state, if SEVONPEND in the System Control Register is set
- a Debug Entry request, if Debug is enabled
- an event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

If the event register is 1, WFE clears it to 0 and returns immediately.

For more information see [Power Management](#), page 45.

Condition flags

This instruction does not change the flags.

Examples

```
WFE ; Wait for event
```

3.6.10.12 WFI

Wait for Interrupt.

3.6.10.12.1 Syntax

`WFI{cond}`

where:

cond is an optional condition code, see [Conditional Execution](#), page 55.

3.6.10.12.2 Operation

WFI is a hint instruction that suspends execution until one of the following events occurs:

- a non-masked interrupt occurs and is taken
- an interrupt masked by PRIMASK becomes pending
- a Debug Entry request.

3.6.10.12.3 Condition Flags

This instruction does not change the flags.

Examples

```
WFI ; Wait for interrupt
```


3.7 Cortex-M3 Processor Peripherals

The following sections are the reference material for the Cortex-M3 processor core peripherals descriptions in this user guide.

3.7.1 About the Cortex-M3 Processor Peripherals

The following table provides the address map of the *Private peripheral bus* (PPB).

Table 39 • Core Peripheral Register Regions

Address	Core Peripheral	See
0xE000E008-0xE000E00F	System control block	Table 50 , page 102
0xE000E010-0xE000E01F	System timer	Table 71 , page 119
0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	Table 40 , page 95
0xE000ED00-0xE000ED3F	System control block	Table 50 , page 102
0xE000ED90-0xE000ED93	MPU Type Register	Reads as zero, indicating no MPU is implemented ¹
0xE000ED90-0xE000EDB8	Memory protection unit	Table 77 , page 123
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	Table 40 , page 95

1. Software can read the MPU Type Register at 0xE000ED90 to test for the presence of a memory protection unit (MPU).

In register descriptions:

- the register *type* is described as follows:
 - RW: Read and write.
 - RO: Read-only.
 - WO: Write-only.
- the *required privilege* gives the privilege level required to access the register, as follows:
 - Privileged: Only privileged software can access the register.
 - Unprivileged: Both unprivileged and privileged software can access the register.

3.7.1.1 Nested Vectored Interrupt Controller

This section describes the Nested Vectored Interrupt Controller (NVIC) and the registers it uses. The NVIC supports:

- 1 to 240 interrupts.
- A programmable priority level of 0-255 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Level and pulse detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external *Non-maskable interrupt* (NMI).

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead. This provides low latency exception handling. The hardware implementation of the NVIC registers is:

Table 40 • NVIC Register Summary

Address	Name	Type	Required privilege	Reset value	See
0xE000E100-0xE000E11C	NVIC_ISER0-NVIC_ISER7	RW	Privileged	0x00000000	Interrupt Set-enable Registers , page 96

Table 40 • NVIC Register Summary (continued)

Address	Name	Type	Required privilege	Reset value	See
0xE000E180-0xE000E19C	NVIC_ICER0-NVIC_ICER7	RW	Privileged	0x00000000	Interrupt Clear-enable Registers , page 97
0xE000E200-0xE000E21C	NVIC_ISPR0-NVIC_ISPR7	RW	Privileged	0x00000000	Interrupt Set-pending Registers , page 97
0xE000E280-0xE000E29C	NVIC_ICPR0-NVIC_ICPR7	RW	Privileged	0x00000000	Interrupt Clear-Pending Registers , page 98
0xE000E300-0xE000E31C	NVIC_IABR0-NVIC_IABR7	RO	Privileged	0x00000000	Interrupt Active Bit Registers , page 98
0xE000E400-0xE000E4EF	NVIC_IPR0-NVIC_IPR59	RW	Privileged	0x00000000	Interrupt Priority Registers , page 99
0xE000EF00	STIR	WO	Configurable ¹	0x00000000	Software Trigger Interrupt Register , page 100

1. See the register description for more information.

3.7.1.2 Accessing the Cortex-M3 Processor NVIC Registers Using CMSIS

CMSIS functions enable software portability between different Cortex-M3 profile processors. To access the NVIC registers when using CMSIS, use the following functions:

Table 41 • CMSIS Access NVIC Functions

CMSIS Function	Description
void NVIC_EnableIRQ(IRQn_Type IRQn) ¹	Enables an interrupt or exception.
void NVIC_DisableIRQ(IRQn_Type IRQn) ^a	Disables an interrupt or exception.
void NVIC_SetPendingIRQ(IRQn_Type IRQn) ^a	Sets the pending status of interrupt or exception to 1.
void NVIC_ClearPendingIRQ(IRQn_Type IRQn) ^a	Clears the pending status of interrupt or exception to 0.
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn) ^a	Reads the pending status of interrupt or exception. This function returns non-zero value if the pending status is set to 1.
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority) ^a	Sets the priority of an interrupt or exception with configurable priority level to 1.
uint32_t NVIC_GetPriority(IRQn_Type IRQn) ^a	Reads the priority of an interrupt or exception with configurable priority level. This function return the current priority level.

1. The input parameter IRQn is the IRQ number, see [Table 22 on page 34](#) for more information.

3.7.1.3 Interrupt Set-enable Registers

The NVIC_ISER0-NVIC_ISER7 registers enable interrupts, and show which interrupts are enabled. See the register summary in [Table 40](#), page 95 for the register attributes.

The bit assignments are:

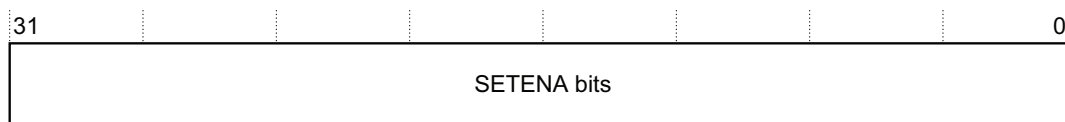
Figure 23 • ISER Register Bit Assignments

Table 42 • NVIC_ISER Bit Assignments

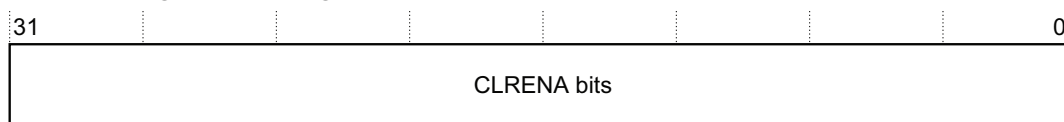
Bits	Name	Function
[31:0]	SETENA	Interrupt set-enable bits. Write: 0: no effect 1: enable interrupt. Read: 0: interrupt disabled 1: interrupt enabled.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

3.7.1.4 Interrupt Clear-enable Registers

The NVIC_ICER0-NVIC_ICER7 registers disable interrupts, and show which interrupts are enabled. See the register summary in [Table 40](#), page 95 for the register attributes.

The bit assignments are:

Figure 24 • ICER Register Bit Assignments**Table 43 • NVIC_ICER Bit Assignments**

Bits	Name	Function
[31:0]	CLRENA	Interrupt clear-enable bits. Write: 0: no effect 1: disable interrupt. Read: 0: interrupt disabled 1: interrupt enabled.

3.7.1.5 Interrupt Set-pending Registers

The NVIC_ISPR0-NVIC_ISPR7 registers force interrupts into the pending state, and show which interrupts are pending. See the register summary in [Table 40](#), page 95 for the register attributes.

The bit assignments are:

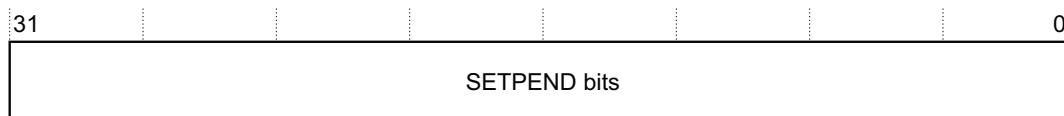
Figure 25 • ISPR Register Bit Assignments

Table 44 • NVIC_ISPR Bit Assignments

Bits	Name	Function
[31:0]	SETPEND	Interrupt set-pending bits. Write: 0: No effect 1: Changes interrupt state to pending. Read: 0: Interrupt is not pending 1: Interrupt is pending.

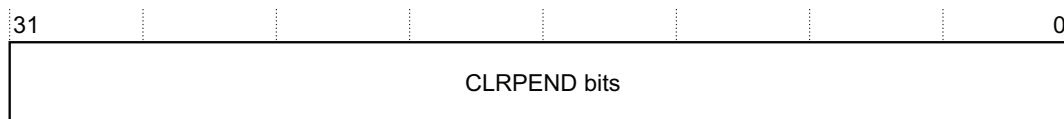
Note: Writing 1 to the NVIC_ISPR bit corresponding to:

- An interrupt that is pending has no effect
- A disabled interrupt sets the state of that interrupt to pending.

3.7.1.6 Interrupt Clear-Pending Registers

The NVIC_ICPR0-NVIC_ICPR7 registers remove the pending state from interrupts, and show which interrupts are pending. See the register summary in [Table 40](#), page 95 for the register attributes.

The bit assignments are:

Figure 26 • ICPR Register Bit Assignments**Table 45 • NVIC_ICPR bit assignments**

Bits	Name	Function
[31:0]	CLRPEND	Interrupt clear-pending bits. Write: 0: no effect 1: removes pending state an interrupt. Read: 0: interrupt is not pending 1: interrupt is pending.

Note: Writing 1 to an NVIC_ICPR bit does not affect the active state of the corresponding interrupt.

3.7.1.7 Interrupt Active Bit Registers

The NVIC_IABR0-NVIC_IABR7 registers indicate which interrupts are active. See the register summary in [Table 40](#), page 95 for the register attributes.

The bit assignments are:

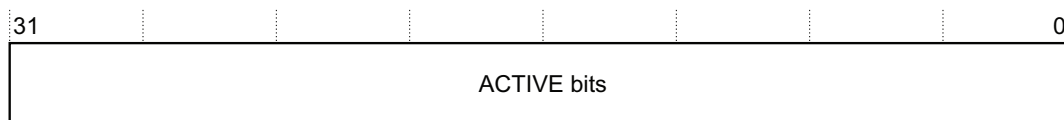
Figure 27 • IABR Register Bit Assignments

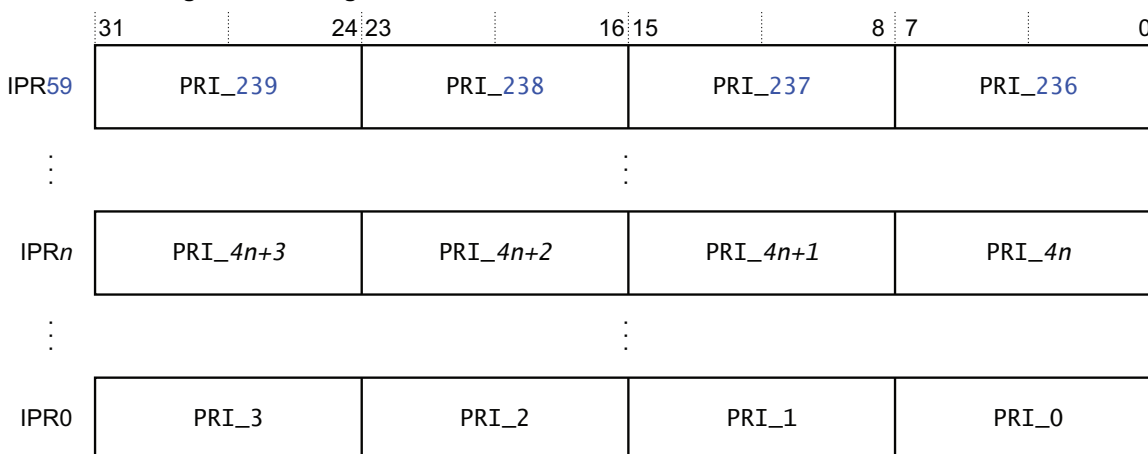
Table 46 • NVIC_IABR Bit Assignments

Bits	Name	Function
[31:0]	ACTIVE	Interrupt active flags: 0: interrupt not active 1: interrupt active.

A bit reads as one if the status of the corresponding interrupt is active or active and pending.

3.7.1.8 Interrupt Priority Registers

The NVIC_IPR0-NVIC_IPR59 registers provide an 8-bit priority field for each interrupt. These registers are byte-accessible. See the register summary in [Table 40](#), page 95 for their attributes. Each register holds four priority fields as shown:

Figure 28 • IPR Register Bit Assignments**Table 47 • NVIC_IPR Bit Assignments**

Bits	Name	Function
[31:24]	Priority, byte offset 3	Each priority field holds a priority value, 0-255. The lower the value, the greater the priority of the corresponding interrupt. The processor implements only bits [7:n] of each field, bits [n-1:0] read as zero and ignore writes.
[23:16]	Priority, byte offset 2	
[15:8]	Priority, byte offset 1	
[7:0]	Priority, byte offset 0	

See [Accessing the Cortex-M3 Processor NVIC Registers Using CMSIS](#), page 96 for more information about the access to the interrupt priority array, which provides the software view of the interrupt priorities.

Find the IPR number and byte offset for interrupt m as follows:

- the corresponding IPR n number (see the preceding table), n is given by $n = m \text{ DIV } 4$
- the byte offset of the required Priority field in this register is $m \text{ MOD } 4$, where:
 - byte offset 0 refers to register bits [7:0]
 - byte offset 1 refers to register bits [15:8]
 - byte offset 2 refers to register bits [23:16]
 - byte offset 3 refers to register bits [31:24].

3.7.1.9 Software Trigger Interrupt Register

Write to the STIR to generate an interrupt from software. See the register summary in [Table 40](#), page 95 for the STIR attributes.

When the USERSETMPEND bit in the SCR is set to 1, unprivileged software can access the STIR, see [System Control Register](#), page 108.

Note: Only privileged software can enable unprivileged access to the STIR.

The bit assignments are:

Figure 29 • IABR Register Bit Assignments

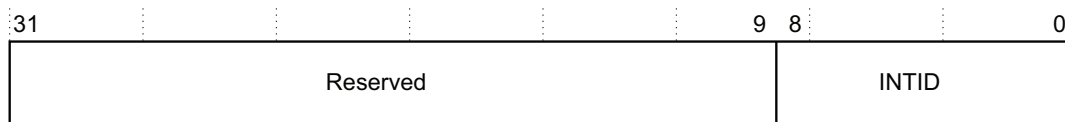


Table 48 • STIR Bit Assignments

Bits	Field	Function
[31:9]		Reserved.
[8:0]	INTID	Interrupt ID of the interrupt to trigger, in the range 0-239. For example, a value of 0x03 specifies interrupt IRQ3.

3.7.1.10 Level-sensitive and Pulse Interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral de-asserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt, see [Hardware and Software Control of Interrupts](#), page 100. For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. This means that the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

See <reference required> for details of which interrupts are level-sensitive and which are pulsed.

3.7.1.10.1 Hardware and Software Control of Interrupts

The Cortex-M3 processor latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- the NVIC detects that the interrupt signal is HIGH and the interrupt is not active
- the NVIC detects a rising edge on the interrupt signal
- software writes to the corresponding interrupt set-pending register bit, see [Interrupt Set-pending Registers](#), page 97, or to the STIR to make an interrupt pending, see [Software Trigger Interrupt Register](#), page 100.

A pending interrupt remains pending until one of the following occurs:

- The processor enters the ISR for the interrupt. This changes the state of the interrupt from pending to active. Then:
 - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.

- For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.

If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.

- Software writes to the corresponding interrupt clear-pending register bit.
For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

For a pulse interrupt, state of the interrupt changes to:

- inactive, if the state was pending
- active, if the state was active and pending.

3.7.1.11 NVIC Design Hints and Tips

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers. See the individual register descriptions for the supported access sizes.

An interrupt can enter pending state even if it is disabled. Disabling an interrupt only prevents the processor from taking that interrupt.

Before programming VTOR to relocate the vector table, ensure the vector table entries of the new vector table are setup for fault handlers, NMI and all enabled exception like interrupts. For more information see [Vector Table Offset Register](#), page 106.

3.7.1.11.1 NVIC programming hints

Software uses the CPSIE I and CPSID I instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including those listed in the following table.

Table 49 • CMSIS Functions for NVIC Control

CMSIS interrupt control function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true (IRQ-Number) if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive (IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

The input parameter IRQn is the IRQ number, see [Properties of the Different Exception Types](#), page 39. For more information about these functions see the CMSIS documentation.

3.7.2 System Control Block

The *System control block* (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The following table lists the SCB registers.

Table 50 • Summary of the System Control Block Registers

Address	Name	Type	Required privilege	Reset value	See
0xE000E008	ACTLR	RW	Privileged	0x00000000	Auxiliary Control Register , page 102
0xE000ED00	CPUID	RO	Privileged	0x412FC230	CPUID Base Register , page 103
0xE000ED04	ICSR	RW ¹	Privileged	0x00000000	Interrupt Control and State Register , page 104
0xE000ED08	VTOR	RW	Privileged	0x00000000	Vector Table Offset Register , page 106
0xE000ED0C	AIRCR	RW ^a	Privileged	0xFA050000	Application Interrupt and Reset Control Register , page 106
0xE000ED10	SCR	RW	Privileged	0x00000000	System Control Register , page 108
0xE000ED14	CCR	RW	Privileged	0x00000200	Configuration and Control Register , page 109
0xE000ED18	SHPR1	RW	Privileged	0x00000000	System Handler Priority Registers , page 110
0xE000ED1C	SHPR2	RW	Privileged	0x00000000	System Handler Priority Registers , page 110
0xE000ED20	SHPR3	RW	Privileged	0x00000000	System Handler Priority Registers , page 110
0xE000ED24	SHCRS	RW	Privileged	0x00000000	System Handler Control and State Register , page 112
0xE000ED28	CFSR	RW	Privileged	0x00000000	Configurable Fault Status Register , page 113
0xE000ED28	MMSR ²	RW	Privileged	0x00	MemManage Fault Status Register , page 114
0xE000ED29	BFSR ^b	RW	Privileged	0x00	BusFault Status Register , page 115
0xE000ED2A	UFSR ^b	RW	Privileged	0x0000	UsageFault Status Register , page 116
0xE000ED2C	HFSR	RW	Privileged	0x00000000	BusFault Status Register , page 115
0xE000ED34	MMAR	RW	Privileged	Unknown	UsageFault Status Register , page 116
0xE000ED38	BFAR	RW	Privileged	Unknown	HardFault Status Register , page 117
0xE000ED3C	AFSR	RW	Privileged	0x00000000	MemManage Fault Address Register , page 118

1. See the register description for more information.

2. A sub-register of the CFSR.

3.7.2.1 Auxiliary Control Register

The ACTLR provides disable bits for the following processor functions:

- IT folding
- write buffer use for accesses to the default memory map
- interruption of multi-cycle instructions.

By default this register is set to provide optimum performance from the Cortex-M3 processor, and does not normally require modification.

See the register summary in the preceding table for the ACTLR attributes. The bit assignments are:

Figure 30 • ACTLR Bit Assignments

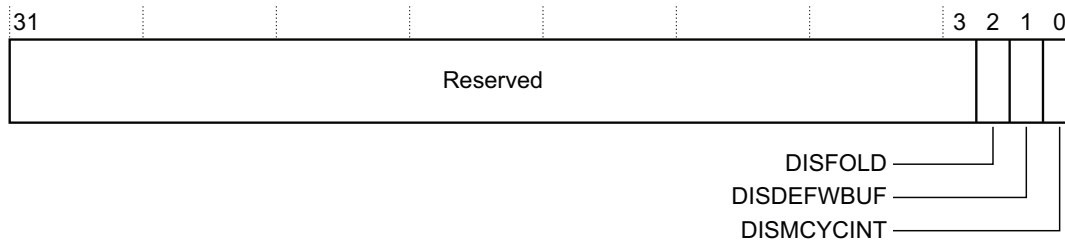


Table 51 • ACTLR Bit Assignments

Bits	Name	Function
[31:3]		Reserved
[2]	DISFOLD	When set to 1, disables the ability of the Cortex-M3 processor to execute an IT instruction in parallel with a neighboring instruction.
[1]	DISDEFWBUF	When set to 1, disables write buffer use during default memory map accesses. This causes all BusFaults to be precise BusFaults but decreases performance because any store to memory must complete before the processor can execute the next instruction. This bit only affects write buffers implemented in the Cortex-M3 processor.
[0]	DISMCYCINT	When set to 1, disables interruption of load multiple and store multiple instructions. This increases the interrupt latency of the processor because any LDM or STM must complete before the processor can stack the current state and enter the interrupt handler.

3.7.2.2 CPUID Base Register

The CPUID register contains the processor part number, version, and implementation information. See the register summary in [Table 50](#), page 102 for its attributes. The bit assignments are:

Figure 31 • CPUID Register Bit Assignments

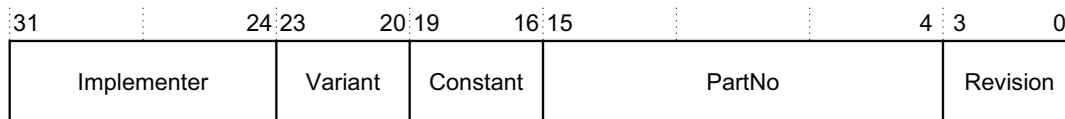


Table 52 • CPUID register Bit Assignments

Bits	Name	Function
[31:24]	Implementer	Implementer code: 0x41 = ARM
[23:20]	Variant	Variant number, the r value in the <i>rnpn</i> product revision identifier: 0x2 = Revision 2
[19:16]	Constant	Reads as 0xF
[15:4]	PartNo	Part number of the processor: 0xC23 = Cortex-M3

Table 52 • CPUID register Bit Assignments (continued)

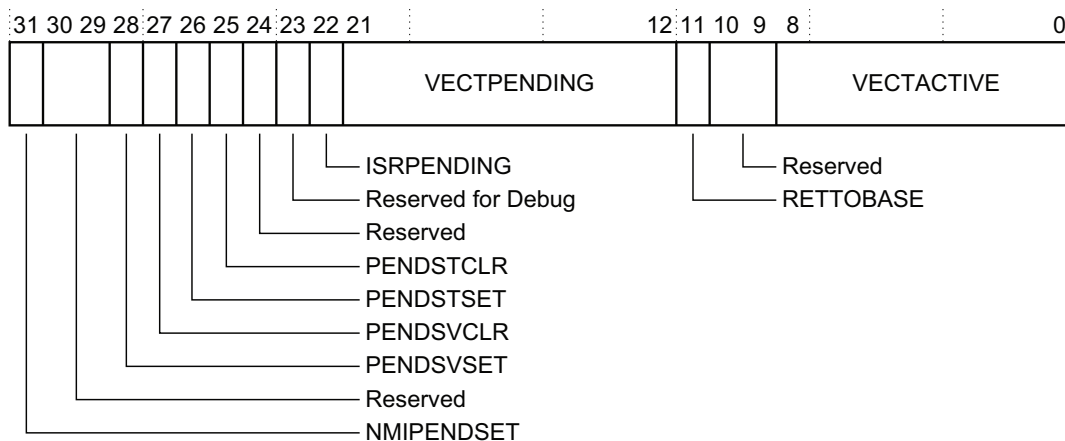
Bits	Name	Function
[3:0]	Revision	Revision number, the p value in the <i>rnppn</i> product revision identifier: 0x0 = Patch 0

3.7.2.3 Interrupt Control and State Register

The ICSR:

- Provides:
 - a set-pending bit for the *Non-Maskable Interrupt* (NMI) exception
 - set-pending and clear-pending bits for the PendSV and SysTick exceptions
- Indicates:
 - the exception number of the exception being processed
 - whether there are preempted active exceptions
 - the exception number of the highest priority pending exception
 - whether any interrupts are pending.

See the register summary in [Table 50](#), page 102, and the Type descriptions in the following table, for the ICSR attributes. The bit assignments are:

Figure 32 • ICSR Bit Assignments**Table 53 • ICSR Bit Assignments**

Bits	Name	Type	Function
[31]	NMIPENDSET	RW	NMI set-pending bit. Write: 0: no effect 1: changes NMI exception state to pending. Read: 0: NMI exception is not pending 1: NMI exception is pending. Because NMI is the highest-priority exception, normally the processor enter the NMI exception handler as soon as it registers a write of 1 to this bit, and entering the handler clears this bit to 0. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler.
[30:29]			Reserved.

Table 53 • ICSR Bit Assignments (continued)

Bits	Name	Type	Function
[28]	PENDSVSET	RW	PendSV set-pending bit. Write: 0: no effect 1: changes PendSV exception state to pending. Read: 0: PendSV exception is not pending 1: PendSV exception is pending. Writing 1 to this bit is the only way to set the PendSV exception state to pending.
[27]	PENDSVCLR	WO	PendSV clear-pending bit. Write: 0: no effect 1: removes the pending state from the PendSV exception.
[26]	PENDSTSET	RW	SysTick exception set-pending bit. Write: 0: no effect 1: changes SysTick exception state to pending. Read: 0: SysTick exception is not pending 1: SysTick exception is pending.
[25]	PENDSTCLR	WO	SysTick exception clear-pending bit. Write: 0: no effect 1: removes the pending state from the SysTick exception. This bit is WO. On a register read its value is Unknown.
[24]			Reserved.
[23]	Reserved for Debug use	RO	This bit is reserved for Debug use and reads-as-zero when the processor is not in Debug.
[22]	ISR_PENDING	RO	Interrupt pending flag, excluding NMI and Faults: 0: interrupt not pending 1: interrupt pending.
[21:18]			Reserved.
[17:12]	VECTPENDING	RO	Indicates the exception number of the highest priority pending enabled exception: 0: no pending exceptions Nonzero: the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the BASEPRI and FAULTMASK registers, but not any effect of the PRIMASK register.
[11]	RETTOBASE	RO	Indicates whether there are preempted active exceptions: 0: there are preempted active exceptions to execute 1: there are no active exceptions, or the currently-executing exception is the only active exception.
[10:9]			Reserved.
[8:0]	VECTACTIVE ^a	RO	Contains the active exception number: 0: Thread mode Nonzero: The exception number ¹ of the currently active exception. Subtract 16 from this value to obtain the CMSIS IRQ number required to index into the Interrupt Clear-Enable, Set-Enable, Clear-Pending, Set-Pending, or Priority Registers, see Table 11 , page 23.

1. This is the same value as IPSR bits[8:0], see [Interrupt Program Status Register](#), page 23.

When you write to the ICSR, the effect is Unpredictable if you:

- write 1 to the PENDSVSET bit and write 1 to the PENDSVCLR bit
- write 1 to the PENDSTSET bit and write 1 to the PENDSTCLR bit.

3.7.2.4 Vector Table Offset Register

The VTOR indicates the offset of the vector table base address from memory address 0x00000000. See the register summary in [Table 50](#), page 102 for its attributes.

The bit assignments are:

Figure 33 • VTOR Bit Assignments

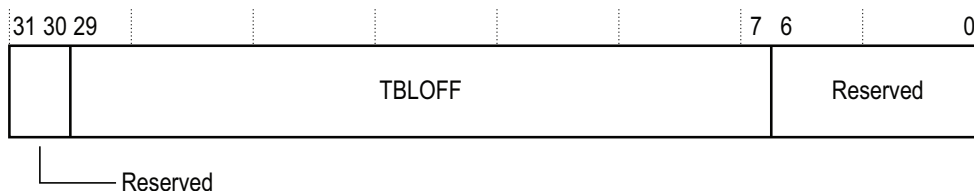


Table 54 • VTOR Bit Assignments

Bits	Name	Function
[31:30]		Reserved.
[29:7]	TBLOFF	Vector table base offset field. It contains bits[29:7] of the offset of the table base from the bottom of the memory map. Bit [29] determines whether the vector table is in the code or SRAM memory region: 0: Code 1: SRAM. Bit [29] is sometimes called the TBLBASE bit.
[6:0]		Reserved.

When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. <Configure the next statement to give the information required for your implementation, the statement reminds you of how to determine the alignment requirement.> The minimum alignment is 32 words, enough for up to 16 interrupts. For more interrupts, adjust the alignment by rounding up to the next power of two. For example, if you require 21 interrupts, the alignment must be on a 64-word boundary because the required table size is 37 words, and the next power of two is 64.

Table alignment requirements mean that bits [6:0] of the table offset are always zero.

3.7.2.5 Application Interrupt and Reset Control Register

The AIRCR provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. See the register summary in [Table 50](#), page 102 and [Table 55](#), page 107 for its attributes.

To write to this register, you must write 0x5FA to the VECTKEY field, otherwise the processor ignores the write.

The bit assignments are:

Figure 34 • AIRCR Bit Assignments

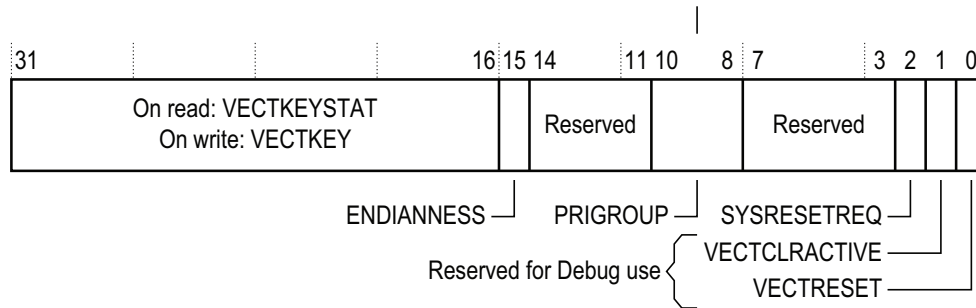


Table 55 • AIRCR Bit Assignments

Bits	Name	Type	Function
[31:16]	Write: VECTKEYSTAT Read: VECTKEY	RW	Register key: Reads as 0xFA05. On writes, write 0x05FA to VECTKEY, otherwise the write is ignored.
[15]	ENDIANNESS	RO	Data endianness bit: 0: Little-endian 1: Big-endian. ENDIANNESS is set from the BIGEND configuration signal during reset.
[14:11]			Reserved
[10:8]	PRIGROUP	R/W	Interrupt priority grouping field. This field determines the split of group priority from subpriority, see " Binary Point " on page 102.
[7:3]			Reserved.
[2]	SYSRESETREQ	WO	System reset request: 0: no system reset request 1: asserts a signal to the outer system that requests a reset. This is intended to force a large system reset of all major components except for debug. This bit reads as 0.
[1]	VECTCLRACTIVE	WO	Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
[0]	VECTRESET	WO	Reserved for Debug use. This bit reads as 0. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.

3.7.2.5.1 Binary Point

The PRIGROUP field indicates the position of the binary point that splits the PRI_n fields in the Interrupt Priority Registers into separate *group priority* and *subpriority* fields. The following table shows how the PRIGROUP value controls this split. <If you implement fewer than 8 priority bits you might require more explanation here, and want to remove invalid rows from the table, and modify the entries in the *number of* columns.>

Table 56 • Priority Grouping

PRIGROUP	Interrupt priority level value, PRI_N[7:0]			Number of	
	Binary point ¹	Group priority bits	Subpriority bits	Group priorities	Subpriorities
b000	bxxxxxx.y	[7:1]	[0]	128	2

Table 56 • Priority Grouping (continued)

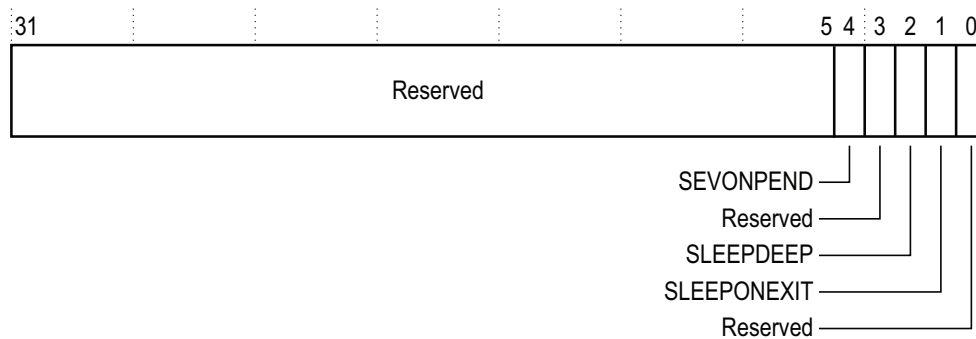
PRIGROUP	Interrupt priority level value, PRI_N[7:0]			Number of	
	Binary point ¹	Group priority bits	Subpriority bits	Group priorities	Subpriorities
b001	bxxxxx.yy	[7:2]	[1:0]	64	4
b010	bxxxxx.yyy	[7:3]	[2:0]	32	8
b011	bxxxx.yyyy	[7:4]	[3:0]	16	16
b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
b110	bx.yyyyyyy	[7]	[6:0]	2	128
b111	b.yyyyyyyy	None	[7:0]	1	256

1. PRI_n[7:0] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

Determining preemption of an exception uses only the group priority field, see [Interrupt Priority Grouping](#), page 41.

3.7.2.6 System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary in [Table 50](#), page 102 for its attributes. The bit assignments are:

Figure 35 • SCR Bit Assignments**Table 57 • SCR Bit Assignments**

Bits	Name	Function
[31:5]		Reserved.
[4]	SEVONPEND	Send Event on Pending bit: 0: only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded 1: enabled events and all interrupts, including disabled interrupts, can wakeup the processor. When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an SEV instruction or an external event.
[3]		Reserved.
[2]	SLEEPDEEP	Controls whether the processor uses sleep or deep sleep as its low power mode: 0: sleep 1: deep sleep

Table 57 • SCR Bit Assignments (continued)

Bits	Name	Function
[1]	SLEEPONEXIT	Indicates sleep-on-exit when returning from Handler mode to Thread mode: 0: do not sleep when returning to Thread mode. 1: enter sleep, or deep sleep, on return from an ISR. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.
[0]		Reserved.

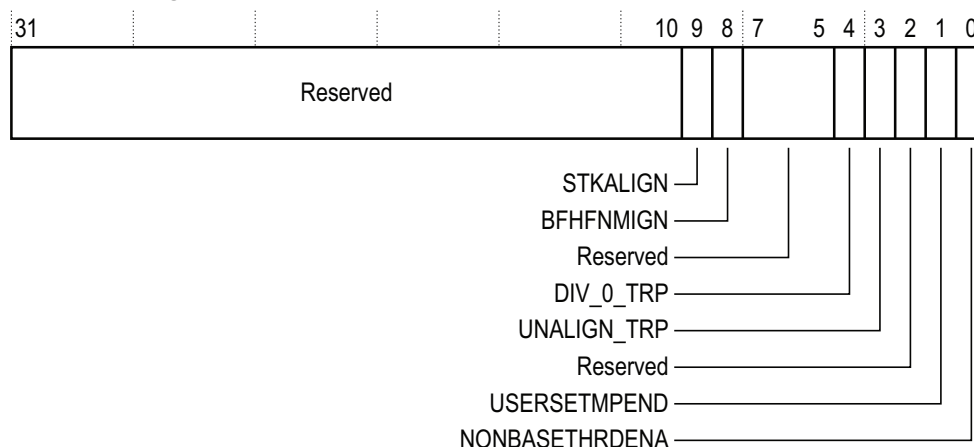
3.7.2.7 Configuration and Control Register

The CCR controls entry to Thread mode and enables:

- the handlers for NMI, HardFault and faults escalated by FAULTMASK to ignore BusFaults
- trapping of divide by zero and unaligned accesses
- access to the STIR by unprivileged software, see ["Software Trigger Interrupt Register" on page 93](#).

See the register summary in [Table 50](#), page 102 for the CCR attributes.

The bit assignments are:

Figure 36 • CCR Bit Assignments**Table 58 • CCR Bit Assignments**

Bits	Name	Function
[31:10]		Reserved.
[9]	STKALIGN	Indicates stack alignment on exception entry: 0: 4-byte aligned 1: 8-byte aligned. On exception entry, the processor uses bit[9] of the stacked PSR to indicate the stack alignment. On return from the exception it uses this stacked bit to restore the correct stack alignment.
[8]	BFHFNMIGN	Enables handlers with priority -1 or -2 to ignore data BusFaults caused by load and store instructions. This applies to the HardFault, NMI, and FAULTMASK escalated handlers: 0: data BusFaults caused by load and store instructions cause a lock-up 1: data BusFaults caused by load and store instructions are ignored. Set this bit to 1 only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect problems.

Table 58 • CCR Bit Assignments (continued)

Bits	Name	Function
[7:5]		Reserved.
[4]	DIV_0_TRP	Enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0: 0: do not trap divide by 0 1: trap divide by 0. When this bit is set to 0, a divide by zero returns a quotient of 0.
[3]	UNALIGN_TRP	Enables unaligned access traps: 0: do not trap unaligned halfword and word accesses 1: trap unaligned halfword and word accesses. If this bit is set to 1, an unaligned access generates a UsageFault. Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of whether UNALIGN_TRP is set to 1.
[2]		Reserved.
[1]	USERSETMPEND	Enables unprivileged software access to the STIR, see Software Trigger Interrupt Register , page 100: 0: disable 1: enable
[0]	NONBASETHRDENA	Indicates how the processor enters Thread mode: 0: processor can enter Thread mode only when no exception is active. 1: processor can enter Thread mode from any level using the appropriate EXC_RETURN value, see Exception Return , page 42.

3.7.2.8 System Handler Priority Registers

The SHPR1-SHPR3 registers set the priority level, 0 to 255 of the exception handlers that have configurable priority.

SHPR1-SHPR3 are byte accessible. See the register summary in [Table 50](#), page 102 for their attributes.

The system fault handlers and the priority field and register for each handler are:

Table 59 • System Fault Handler Priority Fields

Handler	Field	See
MemManage	PRI_4	System Handler Priority Register 2 , page 111
BusFault	PRI_5	
UsageFault	PRI_6	
SVCall	PRI_11	System Handler Priority Register 2 , page 111
PendSV	PRI_14	System Handler Priority Register 2 , page 111
SysTick	PRI_15	

Each PRI_N field is 8 bits wide, but the processor implements only bits [7:M] of each field, and bits [M-1:0] read as zero and ignore writes.

3.7.2.8.1 System Handler Priority Register 1

The bit assignments are:

Figure 37 • SHPR1 Bit Assignments

31	24	23	16	15	8	7	0
Reserved				PRI_6	PRI_5	PRI_4	

Table 60 • SHPR1 Bit Assignments

Bits	Name	Function
[31:24]	PRI_7	Reserved
[23:16]	PRI_6	Priority of system handler 6, UsageFault
[15:8]	PRI_5	Priority of system handler 5, BusFault
[7:0]	PRI_4	Priority of system handler 4, MemManage

3.7.2.8.2 System Handler Priority Register 2

The bit assignments are:

Figure 38 • SHPR2 Bit Assignments

31	24	23	0
PRI_11	Reserved		

Table 61 • SHPR2 Bit Assignments

Bits	Name	Function
[31:24]	PRI_11	Priority of system handler 11, SVCcall
[23:0]		Reserved

3.7.2.8.3 System Handler Priority Register 3

The bit assignments are:

Figure 39 • SHPR3 Bit Assignments

31	24	23	16	15	0
PRI_15	PRI_14	Reserved			

Table 62 • SHPR3 Bit Assignments

Bits	Name	Function
[31:24]	PRI_15	Priority of system handler 15, SysTick exception
[23:16]	PRI_14	Priority of system handler 14, PendSV
[15:0]		Reserved

3.7.2.9 System Handler Control and State Register

The SHCSR enables the system handlers, and indicates:

- the pending status of the BusFault, MemManage fault, and SVC exceptions
- the active status of the system handlers.

See the register summary in [Table 50](#), page 102 for the SHCSR attributes. The bit assignments are:

Figure 40 • SHCSR Bit Assignments

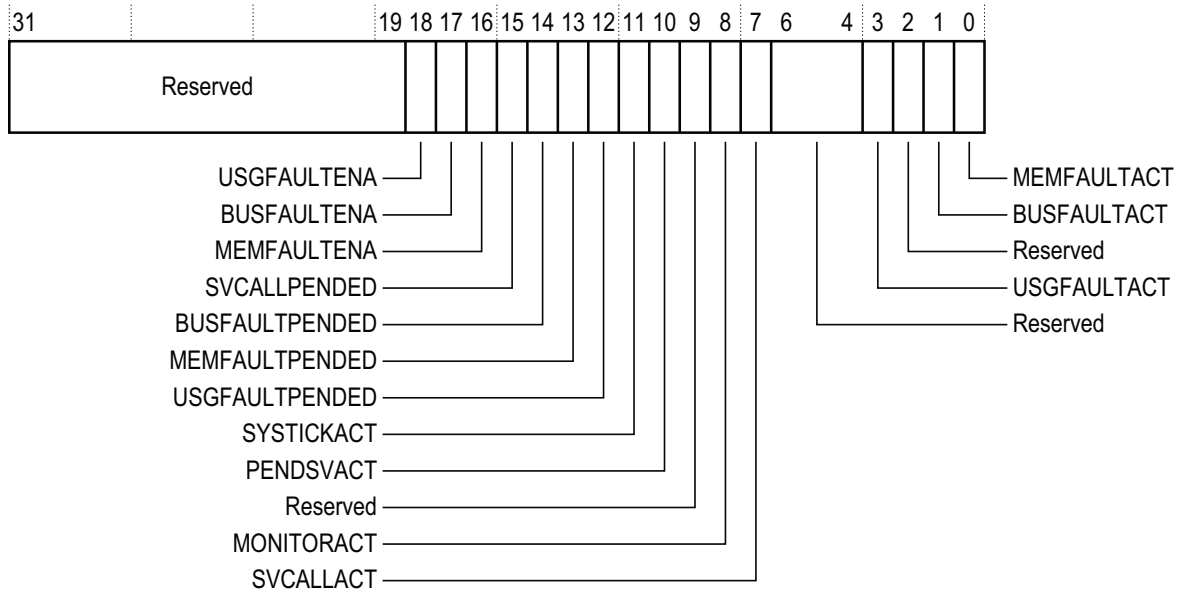


Table 63 • SHCSR Bit Assignments

Bits	Name	Function
[31:19]	-	Reserved
[18]	USGFAULTENA	UsageFault enable bit, set to 1 to enable ¹
[17]	BUSFAULTENA	BusFault enable bit, set to 1 to enable ^a
[16]	MEMFAULTENA	MemManage enable bit, set to 1 to enable ^a
[15]	SVCALLPENDED	SVC call pending bit, reads as 1 if exception is pending ²
[14]	BUSFAULTPENDED	BusFault exception pending bit, reads as 1 if exception is pending ^b
[13]	MEMFAULTPENDED	MemManage fault exception pending bit, reads as 1 if exception is pending ^b
[12]	USGFAULTPENDED	UsageFault exception pending bit, reads as 1 if exception is pending ^b
[11]	SYSTICKACT	SysTick exception active bit, reads as 1 if exception is active ³
[10]	PENDSVACT	PendSV exception active bit, reads as 1 if exception is active
[9]	-	Reserved
[8]	MONITORACT	Debug monitor active bit, reads as 1 if Debug monitor is active
[7]	SVCALLACT	SVC call active bit, reads as 1 if SVC call is active
[6:4]	-	Reserved
[3]	USGFAULTTACT	UsageFault exception active bit, reads as 1 if exception is active
[2]	-	Reserved
[1]	BUSFAULTTACT	BusFault exception active bit, reads as 1 if exception is active

Table 63 • SHCSR Bit Assignments (continued)

Bits	Name	Function
[0]	MEMFAULTACT	MemManage exception active bit, reads as 1 if exception is active

1. Enable bits, set to 1 to enable the exception, or set to 0 to disable the exception.
2. Pending bits, read as 1 if the exception is pending, or as 0 if it is not pending. You can write to these bits to change the pending status of the exceptions.
3. Active bits, read as 1 if the exception is active, or as 0 if it is not active. You can write to these bits to change the active status of the exceptions, but see the Caution in this section.

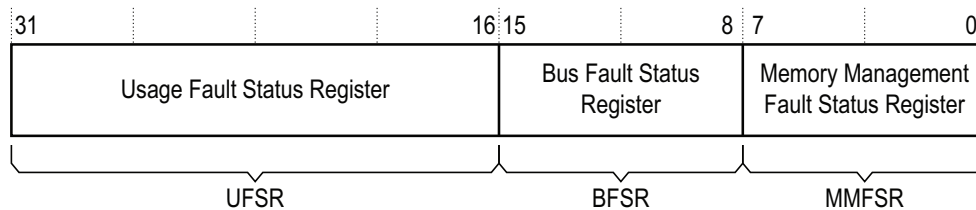
If you disable a system handler and the corresponding fault occurs, the processor treats the fault as a HardFault.

You can write to this register to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

- Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status.
- After you have enabled the system handlers, if you have to change the value of a bit in this register you must use a read-modify-write procedure to ensure that you change only the required bit.

3.7.2.10 Configurable Fault Status Register

The CFSR indicates the cause of a MemManage fault, BusFault, or UsageFault. See the register summary in [Table 50](#), page 102 for its attributes. The bit assignments are:

Figure 41 • CFSR Bit Assignments

The following subsections describe the sub-registers that make up the CFSR:

The CFSR is byte accessible. You can access the CFSR or its sub-registers as follows:

- access the complete CFSR with a word access to 0xE000ED28
- access the MMFSR with a byte access to 0xE000ED28
- access the MMFSR and BFSR with a halfword access to 0xE000ED28
- access the BFSR with a byte access to 0xE000ED29
- access the UFSR with a halfword access to 0xE000ED2A

3.7.2.11 MemManage Fault Status Register

The flags in the MMFSR indicate the cause of memory access faults. The bit assignments are:

Figure 42 • MMFSR Bit Assignments

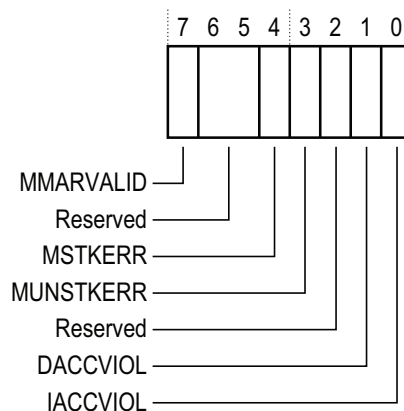


Table 64 • MMFSR Bit Assignments

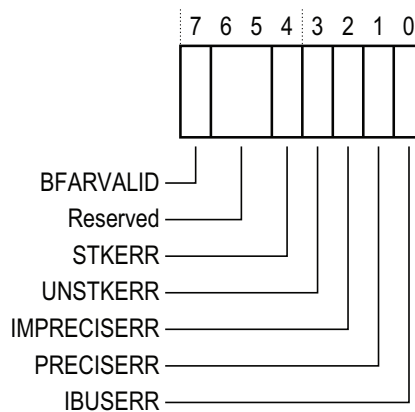
Bits	Name	Function
[7]	MMARVALID	MemManage Fault Address Register (MMFAR) valid flag: 0: value in MMAR is not a valid fault address 1: MMAR holds a valid fault address. If a MemManage fault occurs and is escalated to a HardFault because of priority, the HardFault handler must set this bit to 0. This prevents problems on return to a stacked active MemManage fault handler whose MMAR value has been overwritten.
[6:5]		Reserved.
[4]	MSTKERR	MemManage fault on stacking for exception entry: 0: no stacking fault 1: stacking for an exception entry has caused one or more access violations. When this bit is 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor has not written a fault address to the MMAR.
[3]	MUNSTKERR	MemManage fault on unstacking for a return from exception: 0: no unstacking fault 1: unstack for an exception return has caused one or more access violations. This fault is chained to the handler. This means that when this bit is 1, the original return stack is still present. The processor has not adjusted the SP from the failing return, and has not performed a new save. The processor has not written a fault address to the MMAR.
[2]		Reserved
[1]	DACCVIOL	Data access violation flag: 0: no data access violation fault 1: the processor attempted a load or store at a location that does not permit the operation. When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has loaded the MMAR with the address of the attempted access.

Table 64 • MMFSR Bit Assignments (continued)

Bits	Name	Function
[0]	IACCVIOL	Instruction access violation flag: 0: no instruction access violation fault 1: the processor attempted an instruction fetch from a location that does not permit execution. This fault occurs on any access to an XN region, even when the MPU is disabled or not present. When this bit is 1, the PC value stacked for the exception return points to the faulting instruction. The processor has not written a fault address to the MMAR.

3.7.2.12 BusFault Status Register

The flags in the BFSR indicate the cause of a bus access fault. The bit assignments are:

Figure 43 • MMFSR Bit Assignments**Table 65 • BFSR Bit Assignments**

Bits	Name	Function
[7]	BFARVALID	<i>BusFault Address Register (BFAR)</i> valid flag: 0: value in BFAR is not a valid fault address 1: BFAR holds a valid fault address. The processor sets this bit to 1 after a BusFault where the address is known. Other faults can set this bit to 0, such as a MemManage fault occurring later. If a BusFault occurs and is escalated to a HardFault because of priority, the HardFault handler must set this bit to 0. This prevents problems if returning to a stacked active BusFault handler whose BFAR value has been overwritten.
[6:5]		Reserved.
[4]	STKERR	BusFault on stacking for exception entry: 0: no stacking fault 1: stacking for an exception entry has caused one or more BusFaults. When the processor sets this bit to 1, the SP is still adjusted but the values in the context area on the stack might be incorrect. The processor does not write a fault address to the BFAR.
[3]	UNSTKERR	BusFault on unstacking for a return from exception: 0: no unstacking fault 1: unstack for an exception return has caused one or more BusFaults. This fault is chained to the handler. This means that when the processor sets this bit to 1, the original return stack is still present. The processor does not adjust the SP from the failing return, does not performed a new save, and does not write a fault address to the BFAR.

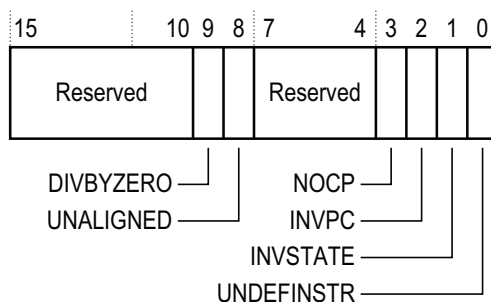
Table 65 • BFSR Bit Assignments (continued)

Bits	Name	Function
[2]	IMPRECISERR	<p>Imprecise data bus error:</p> <p>0: no imprecise data bus error</p> <p>1: a data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.</p> <p>When the processor sets this bit to 1, it does not write a fault address to the BFAR. This is an asynchronous fault. Therefore, if it is detected when the priority of the current process is higher than the BusFault priority, the BusFault becomes pending and becomes active only when the processor returns from all higher priority processes. If a precise fault occurs before the processor enters the handler for the imprecise BusFault, the handler detects both IMPRECISERR set to 1 and one of the precise fault status bits set to 1.</p>
[1]	PRECISERR	<p>Precise data bus error:</p> <p>0: no precise data bus error</p> <p>1: a data bus error has occurred, and the PC value stacked for the exception return points to the instruction that caused the fault.</p> <p>When the processor sets this bit is 1, it writes the faulting address to the BFAR.</p>
[0]	IBUSERR	<p>Instruction bus error:</p> <p>0: no instruction bus error</p> <p>1: instruction bus error.</p> <p>The processor detects the instruction bus error on prefetching an instruction, but it sets the IBUSERR flag to 1 only if it attempts to issue the faulting instruction.</p> <p>When the processor sets this bit is 1, it does not write a fault address to the BFAR.</p>

3.7.2.13 UsageFault Status Register

The UFSR indicates the cause of a UsageFault. The bit assignments are:

Figure 44 • UFSR Bit Assignments

**Table 66 • UFSR Bit Assignments**

Bits	Name	Function
[15:10]		Reserved.
[9]	DIVBYZERO	<p>Divide by zero UsageFault:</p> <p>0: no divide by zero fault, or divide by zero trapping not enabled</p> <p>1: the processor has executed an SDIV or UDIV instruction with a divisor of 0.</p> <p>When the processor sets this bit to 1, the PC value stacked for the exception return points to the instruction that performed the divide by zero.</p> <p>Enable trapping of divide by zero by setting the DIV_0_TRP bit in the CCR to 1, see Configuration and Control Register, page 109.</p>

Table 66 • UFSR Bit Assignments (continued)

Bits	Name	Function
[8]	UNALIGNED	Unaligned access UsageFault: 0: no unaligned access fault, or unaligned access trapping not enabled 1: the processor has made an unaligned memory access. Enable trapping of unaligned accesses by setting the UNALIGN_TRP bit in the CCR to 1, see Configuration and Control Register , page 109. Unaligned LDM, STM, LDRD, and STRD instructions always fault irrespective of the setting of UNALIGN_TRP.
[7:4]		Reserved.
[3]	NOCP	No coprocessor UsageFault. The processor does not support coprocessor instructions: 0: no UsageFault caused by attempting to access a coprocessor 1: the processor has attempted to access a coprocessor.
[2]	INVPC	Invalid PC load UsageFault, caused by an invalid PC load by EXC_RETURN: 0: no invalid PC load UsageFault 1: the processor has attempted an illegal load of EXC_RETURN to the PC, as a result of an invalid context, or an invalid EXC_RETURN value. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that tried to perform the illegal load of the PC.
[1]	INVSTATE	Invalid state UsageFault: 0: no invalid state UsageFault 1: the processor has attempted to execute an instruction that makes illegal use of the EPSR. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that attempted the illegal use of the EPSR. This bit is not set to 1 if an undefined instruction uses the EPSR.
[0]	UNDEFINSTR	Undefined instruction UsageFault: 0: no undefined instruction UsageFault 1: the processor has attempted to execute an undefined instruction. When this bit is set to 1, the PC value stacked for the exception return points to the undefined instruction. An undefined instruction is an instruction that the processor cannot decode.

The UFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

3.7.2.14 HardFault Status Register

The HFSR gives information about events that activate the HardFault handler. See the register summary in [Table 50](#), page 102 for its attributes.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0. The bit assignments are:

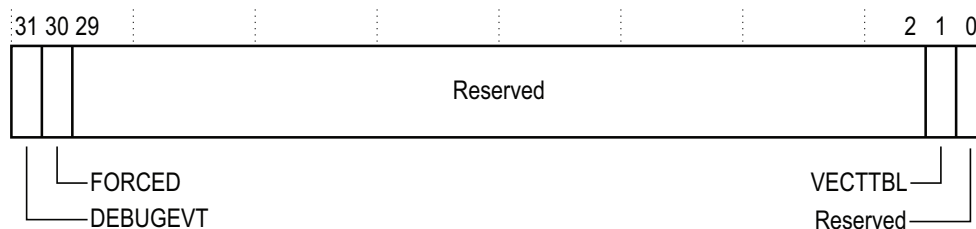
Figure 45 • HFSR Bit Assignments

Table 67 • HFSR Bit Assignments

Bits	Name	Function
[31]	DEBUGEVT	Reserved for Debug use. When writing to the register you must write 0 to this bit, otherwise behavior is Unpredictable.
[30]	FORCED	Indicates a forced HardFault, generated by escalation of a fault with configurable priority that cannot be handles, either because of priority or because it is disabled: 0: no forced HardFault 1: forced HardFault. When this bit is set to 1, the HardFault handler must read the other fault status registers to find the cause of the fault.
[29:2]		Reserved.
[1]	VECTTBL	Indicates a BusFault on a vector table read during exception processing: 0: no BusFault on vector table read 1: BusFault on vector table read. This error is always handled by the HardFault handler. When this bit is set to 1, the PC value stacked for the exception return points to the instruction that was preempted by the exception.
[0]		Reserved.

The HFSR bits are sticky. This means as one or more fault occurs, the associated bits are set to 1. A bit that is set to 1 is cleared to 0 only by writing 1 to that bit, or by a reset.

3.7.2.15 MemManage Fault Address Register

The MMFAR contains the address of the location that generated a MemManage fault. See the register summary in [Table 50](#), page 102 for its attributes. The bit assignments are:

Table 68 • MMFAR Bit Assignments

Bits	Name	Function
[31:0]	ADDRESS	When the MMARVALID bit of the MMFSR is set to 1, this field holds the address of the location that generated the MemManage fault

When an unaligned access faults, the address is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size.

Flags in the MMFSR indicate the cause of the fault, and whether the value in the MMFAR is valid. See [MemManage Fault Status Register](#), page 114.

3.7.2.16 BusFault Address Register

The BFAR contains the address of the location that generated a BusFault. See the register summary in [Table 50](#), page 102 for its attributes. The bit assignments are:

Table 69 • BFAR Bit Assignments

Bits	Name	Function
[31:0]	ADDRESS	When the BFARVALID bit of the BFSR is set to 1, this field holds the address of the location that generated the BusFault

When an unaligned access faults the address in the BFAR is the one requested by the instruction, even if it is not the address of the fault.

Flags in the BFSR indicate the cause of the fault, and whether the value in the BFAR is valid. See [BusFault Status Register](#), page 115.

3.7.2.17 Auxiliary Fault Status Register

The AFSR contains additional system fault information. See the register summary in [Table 50](#), page 102 for its attributes.

This register is read, write to clear. This means that bits in the register read normally, but writing 1 to any bit clears that bit to 0.

The bit assignments are:

Table 70 • AFSR Bit Assignments

Bits	Name	Function
[31:0]	IMPDEF	Implementation defined. The bits map to the AUXFAULT input signals.

Each AFSR bit maps directly to an **AUXFAULT** input of the processor, and a single-cycle HIGH signal on the input sets the corresponding AFSR bit to one. It remains set to 1 until you write 1 to the bit to clear it to zero.

When an AFSR bit is latched as one, an exception does not occur. Use an interrupt if an exception is required.

3.7.2.18 System Control Block Design - Hints and Tips

Ensure software uses aligned accesses of the correct size to access the system control block registers:

- except for the CFSR and SHPR1-SHPR3, it must use aligned word accesses.
- for the CFSR and SHPR1-SHPR3 it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to system control block registers.

In a fault handler, to determine the true faulting address:

- Read and save the MMFAR or BFAR value.
- Read the MMARVALID bit in the MMFSR, or the BFARVALID bit in the BFSR. The MMFAR or BFAR address is valid only if this bit is 1.

Software must follow this sequence because another higher priority exception might change the MMFAR or BFAR value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the MMFAR or BFAR value.

3.7.3 System Timer, SysTick

The processor has a 24-bit system timer, SysTick, that counts down from the reload value to zero, reloads (wraps to) the value in the SYST_RVR register on the next clock edge, then counts down on subsequent clocks.

When the processor is halted for debugging the counter does not decrement.

The system timer registers are:

Table 71 • System Timer Registers Summary

Address	Name	Type	Required Privilege	Reset Value	See
0xE000E010	SYST_CTRL	RW	Privileged	0x00000004	SysTick Control and Status Register , page 120
0xE000E014	SYST_RVR	RW	Privileged	0x00000000	SysTick Reload Value Register , page 120
0xE000E018	SYST_CVR	RW	Privileged	0x00000000	SysTick Current Value Register , page 121
0xE000E01C	SYST_CALIB	RO	Privileged	0xC0000000 ¹	SysTick Calibration Value Register , page 121

1. SysTick calibration value.

3.7.3.1 SysTick Control and Status Register

The SYST_CTRL register enables the SysTick features. See the register summary in the preceding table for its attributes. The bit assignments are:

Figure 46 • SYST_CTRL Register Bit Assignments

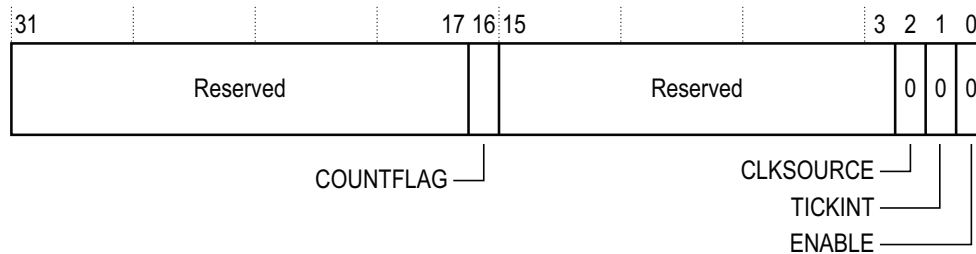


Table 72 • SYST_CTRL Register Bit Assignments

Bits	Name	Function
[31:17]		Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read.
[15:3]		Reserved.
[2]	CLKSOURCE	Selects the SysTick timer clock source: 1: processor clock. Determined by STCLK_DIVISOR bits in M3_CR register.
[1]	TICKINT	Enables SysTick exception request: 0: counting down to zero does not assert the SysTick exception request 1: counting down to zero to asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.
[0]	ENABLE	Enables the counter: 0: counter disabled 1: counter enabled.

When ENABLE is set to 1, the counter loads the RELOAD value from the SYST_RVR register and then counts down. On reaching 0, it sets the COUNTFLAG to 1 and optionally asserts the SysTick depending on the value of TICKINT. It then loads the RELOAD value again, and begins counting.

3.7.3.2 SysTick Reload Value Register

The SYST_RVR register specifies the start value to load into the SYST_CVR register. See the register summary in [Table 71](#), page 119 for its attributes. The bit assignments are:

Figure 47 • SYST_RVR Register Bit Assignments

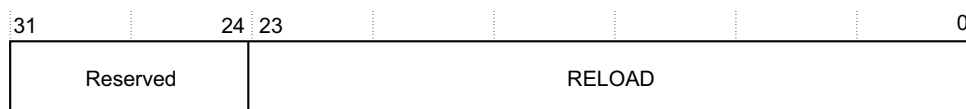


Table 73 • SYST_RVR Register Bit Assignments

Bits	Name	Function
[31:24]		Reserved.

Table 73 • SYST_RVR Register Bit Assignments (continued)

Bits	Name	Function
[23:0]	RELOAD	Value to load into the SYST_CVR register when the counter is enabled and when it reaches 0, see Calculating the RELOAD Value , page 121.

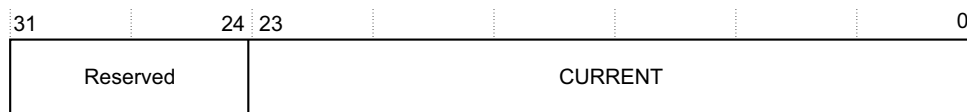
3.7.3.2.1 Calculating the RELOAD Value

The RELOAD value can be any value in the range 0x00000001-0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick exception request and COUNTFLAG are activated when counting from 1 to 0.

The RELOAD value is calculated according to its use. To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

3.7.3.3 SysTick Current Value Register

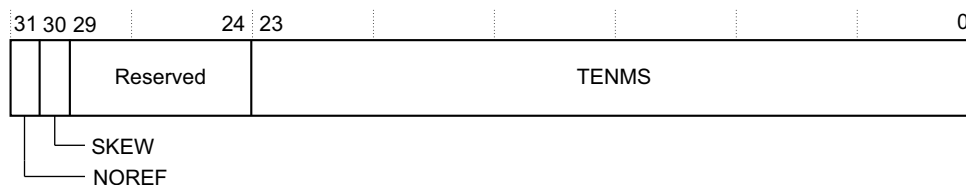
The SYST_CVR register contains the current value of the SysTick counter. See the register summary in [Table 71](#), page 119 for its attributes. The bit assignments are:

Figure 48 • SYST_CVR Register Bit Assignments**Table 74 • SYST_CVR Register Bit Assignments**

Bits	Name	Function
[31:24]		Reserved.
[23:0]	CURRENT	Reads return the current value of the SysTick counter. A write of any value clears the field to 0, and also clears the SysTick CTRL.COUNTFLAG bit to 0.

3.7.3.4 SysTick Calibration Value Register

The SYST_CALIB register indicates the SysTick calibration properties. See the register summary in [Table 71](#), page 119 for its attributes. The bit assignments are:

Figure 49 • SYST_CALIB Register Bit Assignments**Table 75 • SYST_CALIB Register Bit Assignments**

Bits	Name	Function
[31]	NOREF	Reads as one. Indicates that no separate reference clock is provided.
[30]	SKEW	Reads as one. Calibration value for the 10ms inexact timing is not known because TENMS is not known. This can affect the suitability of SysTick as a software real time clock.

Table 75 • SYST_CALIB Register Bit Assignments (continued)

Bits	Name	Function
[29:24]		Reserved.
[23:0]	TENMS	Reads as zero. Indicates calibration value is not known.

If calibration information is not known, calculate the calibration value required from the frequency of the processor clock or external clock.

3.7.3.5 SysTick Design Hints and Tips

The SysTick counter runs on the processor clock. If this clock signal is stopped for Low-power mode, the SysTick counter stops.

Ensure software uses aligned word accesses to access the SysTick registers.

The SysTick counter reload and current value are undefined at reset, the correct initialization sequence for the SysTick counter is:

1. Program reload value.
2. Clear current value.
3. Program Control and Status register.

3.7.4 Memory Protection Unit

This section describes the *Memory protection unit* (MPU).

The MPU divides the memory map into a number of regions, and defines the location, size, access permissions, and memory attributes of each region. It supports:

- independent attribute settings for each region
- overlapping regions
- export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M3 processor MPU defines:

- eight separate memory regions, 0-7
- a background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M3 processor MPU memory map is unified. This means instruction accesses and data accesses have same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a MemManage fault. This causes a fault exception, and might cause termination of the process in an OS environment.

In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

Configuration of MPU regions is based on memory types, see [Memory Regions, Types and Attributes](#), page 29.

The following table shows the possible MPU region attributes. These include shareability and cache behavior attributes that are not relevant to most microcontroller implementations. See [MPU Configuration for a Microcontroller](#), page 132 for guidelines for programming such an implementation.

Table 76 • Memory Attributes Summary

Memory Type	Shareability	Other Attributes	Description
Strongly-ordered			All accesses to Strongly-ordered memory occur in program order. All Strongly-ordered regions are assumed to be shared.
Device	Shared		Memory-mapped peripherals that several processors share.
	Non-shared		Memory-mapped peripherals that only a single processor uses.
Normal	Shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that is shared between several processors.
	Non-shared	Non-cacheable Write-through Cacheable Write-back Cacheable	Normal memory that only a single processor uses.

Use the MPU registers to define the MPU regions and their attributes. The MPU registers are:

Table 77 • MPU Registers Summary

Address	Name	Type	Required privilege	Reset value	See
0xE000ED90	MPU_TYPE	RO	Privileged	0x00000800	MPU Type Register , page 124
0xE000ED94	MPU_CTRL	RW	Privileged	0x00000000	MPU Control Register , page 124
0xE000ED98	MPU_RNR	RW	Privileged	0x00000000	MPU Region Number Register , page 125
0xE000ED9C	MPU_RBAR	RW	Privileged	0x00000000	MPU Region Base Address Register , page 126
0xE000EDA0	MPU_RASR	RW	Privileged	0x00000000	MPU Region Attribute and Size Register , page 127
0xE000EDA4	MPU_RBAR_A1	RW	Privileged	0x00000000	Alias of MPU_RBAR, see MPU Region Base Address Register , page 126
0xE000EDA8	MPU_RASR_A1	RW	Privileged	0x00000000	Alias of MPU_RASR, see MPU Region Attribute and Size Register , page 127
0xE000EDAC	MPU_RBAR_A2	RW	Privileged	0x00000000	Alias of MPU_RBAR, see MPU Region Base Address Register , page 126
0xE000EDB0	MPU_RASR_A2	RW	Privileged	0x00000000	Alias of MPU_RASR, see MPU Region Attribute and Size Register , page 127
0xE000EDB4	MPU_RBAR_A3	RW	Privileged	0x00000000	Alias of MPU_RBAR, see MPU Region Base Address Register , page 126
0xE000EDB8	MPU_RASR_A3	RW	Privileged	0x00000000	Alias of MPU_RASR, see MPU Region Attribute and Size Register , page 127

3.7.4.1 MPU Type Register

The MPU_TYPE register indicates whether the MPU is present, and if so, how many regions it supports. See the register summary in [Table 77](#), page 123 for its attributes. The bit assignments are:

Figure 50 • MPU_TYPE Register Bit Assignments

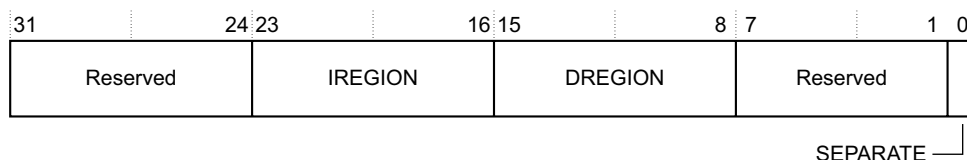


Table 78 • MPU_TYPE Register Bit Assignments

Bits	Name	Function
[31:24]		Reserved.
[23:16]	IREGION	Indicates the number of supported MPU instruction regions. Always contains 0x00. The MPU memory map is unified and is described by the DREGION field.
[15:8]	DREGION	Indicates the number of supported MPU data regions: 0x08=Eight MPU regions.
[7:0]		Reserved.
[0]	SEPARATE	Indicates support for unified or separate instruction and data memory maps: 0: unified

3.7.4.2 MPU Control Register

The MPU_CTRL register:

- enables the MPU
- enables the default memory map background region
- enables use of the MPU when in the HardFault, *Non-maskable Interrupt* (NMI), and FAULTMASK escalated handlers.

See the register summary in [Table 77](#), page 123 for the MPU_CTRL attributes. The bit assignments are:

Figure 51 • MPU_CTRL Register Bit Assignments (continued)

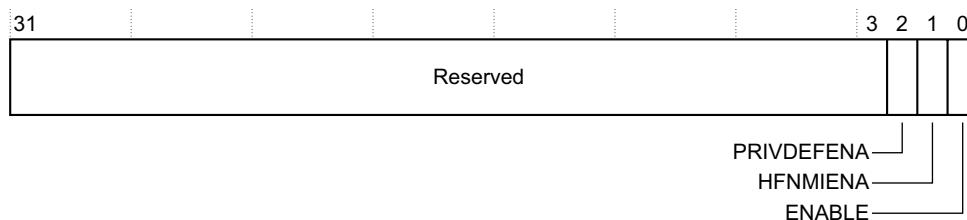


Table 79 • MPU_CTRL Register Bit Assignments

Bits	Name	Function
[31:3]		Reserved.

Table 79 • MPU_CTRL Register Bit Assignments (continued)

Bits	Name	Function
[2]	PRIVDEFENA	<p>Enables privileged software access to the default memory map:</p> <p>0: If the MPU is enabled, disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault.</p> <p>1: If the MPU is enabled, enables use of the default memory map as a background region for privileged software accesses.</p> <p>When enabled, the background region acts as if it is region number -1. Any region that is defined and enabled has priority over this default map.</p> <p>If the MPU is disabled, the processor ignores this bit.</p>
[1]	HFNMENA	<p>Enables the operation of MPU during HardFault, NMI, and FAULTMASK handlers.</p> <p>When the MPU is enabled:</p> <p>0: MPU is disabled during HardFault, NMI, and FAULTMASK handlers, regardless of the value of the ENABLE bit</p> <p>1: the MPU is enabled during HardFault, NMI, and FAULTMASK handlers.</p> <p>When the MPU is disabled, if this bit is set to 1 the behavior is Unpredictable.</p>
[0]	ENABLE	<p>Enables the MPU:</p> <p>0: MPU disabled</p> <p>1: MPU enabled</p>

When ENABLE and PRIVDEFENA are both set to 1:

- For privileged accesses, the *default memory map* is as described in ["Memory Model" on page 25](#). Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a MemManage fault.

XN and Strongly-ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set to 1, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFENA bit is set to 1. If the PRIVDEFENA bit is set to 1 and no regions are enabled, then only privileged software can operate.

When the ENABLE bit is set to 0, the system uses the default memory map. This has the same memory attributes as if the MPU is not implemented, see [Table 17](#), page 30. The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFENA is set to 1.

Unless HFNMIENA is set to 1, the MPU is not enabled when the processor is executing the handler for an exception with priority -1 or -2. These priorities are only possible when handling a HardFault or NMI exception, or when FAULTMASK is enabled. Setting the HFNMIENA bit to 1 enables the MPU when operating with these two priorities.

3.7.4.3 MPU Region Number Register

The MPU_RNR selects which memory region is referenced by the MPU_RBAR and MPU_RASR registers. See the register summary in [Table 77](#), page 123 for its attributes. The bit assignments are:

Figure 52 • SYST_CVR Register Bit Assignments

[illegible]

Table 80 • MPU_RNR Bit Assignments

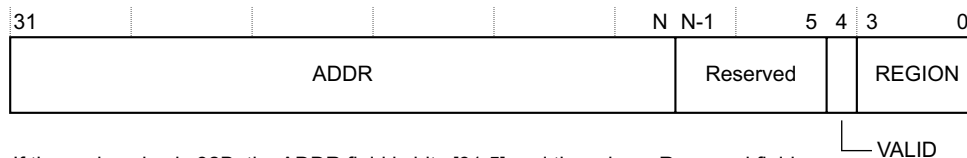
Bits	Name	Function
[31:8]		Reserved.
[7:0]	REGION	Indicates the MPU region referenced by the MPU_RBAR and MPU_RASR registers. The MPU supports 8 memory regions, so the permitted values of this field are 0-7.

Normally, you write the required region number to this register before accessing the MPU_RBAR or MPU_RASR. However you can change the region number by writing to the MPU_RBAR with the VALID bit set to 1, see [MPU Region Base Address Register](#), page 126. This write updates the value of the REGION field.

3.7.4.4 MPU Region Base Address Register

The MPU_RBAR defines the base address of the MPU region selected by the MPU_RNR, and can update the value of the MPU_RNR. See the register summary in [Table 77](#), page 123 for its attributes.

Write MPU_RBAR with the VALID bit set to 1 to change the current region number and update the MPU_RNR. The bit assignments are:

Figure 53 • MPU_RBAR Bit Assignments

If the region size is 32B, the ADDR field is bits [31:5] and there is no Reserved field

Table 81 • MPU_RBAR Bit Assignments

Bits	Name	Function
[31:N]	ADDR	Region base address field. The value of N depends on the region size. For more information see ADDR Field , page 126.
[(N-1):5]		Reserved.
[4]	VALID	MPU Region Number valid bit: Write: 0 = MPU_RNR not changed, and the processor: updates the base address for the region specified in the MPU_RNR ignores the value of the REGION field 1 = the processor: updates the value of the MPU_RNR to the value of the REGION field updates the base address for the region specified in the REGION field. Always reads as zero.
[3:0]	REGION	MPU region field: For the behavior on writes, see the description of the VALID field. On reads, returns the current region number, as specified by the MPU_RNR.

3.7.4.4.1 ADDR Field

The ADDR field is bits[31:N] of the MPU_RBAR. The region size, as specified by the SIZE field in the MPU_RASR, defines the value of N:

$$N = \text{Log}_2(\text{Region size in bytes}),$$

If the region size is configured to 4GB, in the MPU_RASR, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x00000000.

The base address is aligned to the size of the region. For example, a 64KB region must be aligned on a multiple of 64KB, for example, at 0x00010000 or 0x00020000.

3.7.4.5 MPU Region Attribute and Size Register

The MPU_RASR defines the region size and memory attributes of the MPU region specified by the MPU_RNR, and enables that region and any subregions. See the register summary in [Table 77](#), page 123 for its attributes.

MPU_RASR is accessible using word or halfword accesses:

- the most significant halfword holds the region attributes
- the least significant halfword holds the region size and the region and subregion enable bits.

The bit assignments are:

Figure 54 • MPU_RASR Bit Assignments

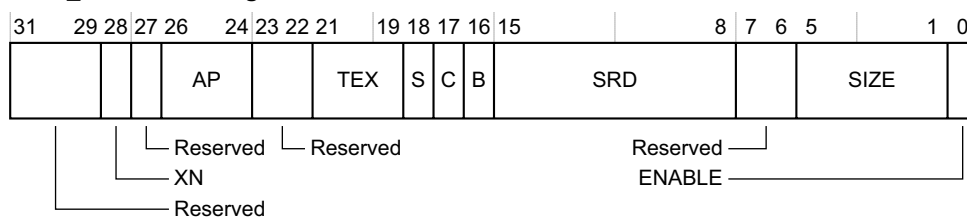


Table 82 • MPU_RASR Bit Assignments

Bits	Name	Function
[31:29]		Reserved.
[28]	XN	Instruction access disable bit: 0: instruction fetches enabled 1: instruction fetches disabled.
[27]		Reserved.
[26:24]	AP	Access permission field, see Table 86 , page 129.
[23:22]		Reserved.
[21:19, 17, 16]	TEX, C, B	Memory access attributes, see Table 84 , page 128.
[18]	S	Shareable bit, see Table 84 , page 128.
[15:8]	SRD	Subregion disable bits. For each bit in this field: 0: corresponding sub-region is enabled 1: corresponding sub-region is disabled See Subregions , page 131 for more information. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as 0x00.
[7:6]		Reserved.
[5:1]	SIZE	Specifies the size of the MPU protection region. The minimum permitted value is 3 (b00010), see SIZE Field Values , page 128 for more information.
[0]	ENABLE	Region enable bit.

For information about access permission, refer to [MPU Access Permission Attributes](#), page 128.

3.7.4.5.1 SIZE Field Values

The SIZE field defines the size of the MPU memory region specified by the MPU_RNR. as follows:

$$(\text{Region size in bytes}) = 2(\text{SIZE}+1)$$

The smallest permitted region size is 32B, corresponding to a SIZE value of 4. The following table provides example SIZE values, with the corresponding region size and value of N in the MPU_RBAR.

Table 83 • Example SIZE Field Values

SIZE value	Region size	Value of N ¹	Note
b00100 (4)	32B	5	Minimum permitted size
b01001 (9)	1KB	10	
b10011 (19)	1MB	20	
b11101 (29)	1GB	30	
b11111 (31)	4GB	32	Maximum possible size

1. In the MPU_RBAR, see [MPU Region Base Address Register](#), page 126.

3.7.4.6 MPU Access Permission Attributes

This section describes the MPU access permission attributes. The access permission bits, TEX, C, B, S, AP, and XN, of the MPU_RASR, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The following table shows the encodings for the TEX, C, B, and S access permission bits.

Table 84 • TEX, C, B, and S Encoding

TEX	C	B	S	Memory type	Shareability	Other attributes
b000	0	0	x ¹	Strongly-ordered	Shareable	
		1	x ^a	Device	Shareable	
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner write-back. No write allocate.
			1		Shareable	
b001	0	0	0	Normal	Not shareable	Outer and inner non-cacheable.
			1		Shareable	
		1	x ^a	Reserved encoding		
	1	0	x ^a	Implementation defined attributes.		
			0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
		1	1		Shareable	
b010	0	0	x ^a	Device	Not shareable	Nonshared Device.
		1	x ^a	Reserved encoding		
	1	x ^a	x ^a	Reserved encoding		
b1BB	A	A	0	Normal	Not shareable	Cached memory ² , BB = outer policy, AA = inner policy.
			1		Shareable	

1. The MPU ignores the value of this bit.

2. See [Table 85](#), page 129 for the encoding of the AA and BB bits.

The following table describes the cache policy for memory attribute encodings with a TEX value is in the range 4-7.

Table 85 • Cache Policy for Memory Attribute Encoding

Encoding, AA or BB	Corresponding cache policy
00	Non-cacheable
01	Write back, write and read allocate
10	Write through, no write allocate
11	Write back, no write allocate

The following table lists the AP encodings that define the access permissions for privileged and unprivileged software.

Table 86 • AP Encoding

AP[2:0]	Privileged Permissions	Unprivileged Permissions	Description
000	No access	No access	All accesses generate a permission fault
001	RW	No access	Access from privileged software only
010	RW	RO	Writes by unprivileged software generate a permission fault
011	RW	RW	Full access
100	Unpredictable	Unpredictable	Reserved
101	RO	No access	Reads by privileged software only
110	RO	RO	Read only, by privileged or unprivileged software
111	RO	RO	Read only, by privileged or unprivileged software

3.7.4.7 MPU Mismatch

When an access violates the MPU permissions, the processor generates a MemManage fault, see [Exceptions and Interrupts](#), page 28. The MMFSR indicates the cause of the fault. See [Auxiliary Fault Status Register](#), page 119 for more information.

3.7.4.8 Updating an MPU Region

To update the attributes for an MPU region, update the MPU_RNR, MPU_RBAR and MPU_RASR registers. You can program each register separately, or use a multiple-word write to program all of these registers. You can use the MPU_RBAR and MPU_RASR aliases to program up to four regions simultaneously using an STM instruction.

Updating an MPU region using separate words

Simple code to configure one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address

LDR R0,=MPU_RNR          ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]        ; Region Number
STR R4, [R0, #0x4]        ; Region Base Address
```

```

STRH R2, [R0, #0x8]      ; Region Size and Enable
STRH R3, [R0, #0xA]      ; Region Attribute

```

Disable a region before writing new region settings to the MPU if you have previously enabled the region being changed. For example:

```

; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address

LDR R0, =MPU_RNR          ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]        ; Region Number
BIC R2, R2, #1            ; Disable
STRH R2, [R0, #0x8]       ; Region Size and Enable
STR R4, [R0, #0x4]        ; Region Base Address
STRH R3, [R0, #0xA]       ; Region Attribute
ORR R2, #1                ; Enable
STRH R2, [R0, #0x8]       ; Region Size and Enable

```

Software must use memory barrier instructions:

- before MPU setup if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings
- after MPU setup if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanism cause memory barrier behavior.

Software does not need any memory barrier instructions during MPU setup, because it accesses the MPU through the PPB, which is a Strongly-Ordered memory region.

For example, if you want all of the memory access behavior to take effect immediately after the programming sequence, use a DSB instruction and an ISB instruction. A DSB is required after changing MPU settings, such as at the end of context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then you do not require an ISB.

Updating an MPU region using multi-word writes

You can program directly using multi-word writes, depending on how the information is divided. Consider the following reprogramming:

```

; R1 = region number
; R2 = address
; R3 = size, attributes in one

LDR R0, =MPU_RNR          ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]        ; Region Number
STR R2, [R0, #0x4]        ; Region Base Address
STR R3, [R0, #0x8]        ; Region Attribute, Size and Enable

```

You can do this in two words for pre-packed information. This means that the MPU_RBAR contains the required region number and had the VALID bit set to 1, see [MPU Region Base Address Register](#), page 126. Use this when the data is statically packed, for example in a boot loader:

```

; R1 = address and region number in one
; R2 = size and attributes in one

LDR R0, =MPU_MPU_RBAR    ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]    ; Region base address and
                        ; region number combined with VALID (bit 4) set to 1
STR R2, [R0, #0x4]    ; Region Attribute, Size and Enable

```

3.7.4.8.1 Subregions

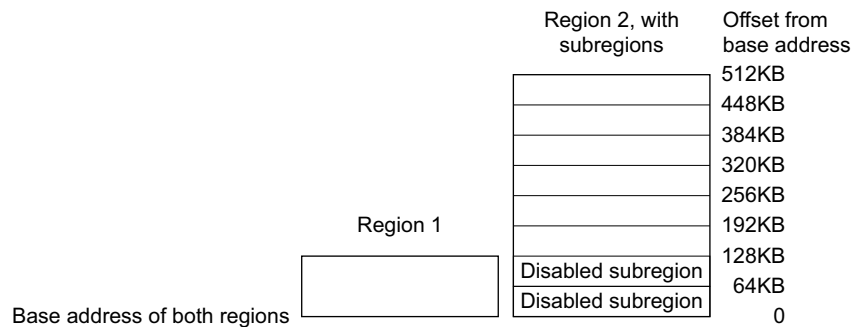
Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the MPU_RASR to disable a subregion, see [MPU Region Attribute and Size Register](#), page 127. The least significant bit of SRD controls the first subregion, and the most significant bit controls the last subregion. Disabling a subregion means another region overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions. With regions of these sizes, you must set the SRD field to 0x00, otherwise the MPU behavior is Unpredictable.

Example of SRD use

Two regions with the same base address overlap. Region one is 128KB, and region two is 512KB. To ensure the attributes from region one apply to the first 128KB region, set the SRD field for region two to b00000011 to disable the first two subregions, as shown in the following figure.

Figure 55 • SRD Field



3.7.4.9 MPU Design Hints and Tips

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure software uses aligned accesses of the correct size to access MPU registers:

- except for the MPU_RASR, it must use aligned word accesses
- for the MPU_RASR it can use byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

3.7.4.9.1 MPU Configuration for a Microcontroller

Usually, a microcontroller system has only a single processor and no caches. In such a system, program the MPU as indicated in the following table.

Table 87 • Memory Region Attributes for a Microcontroller

Memory region	TEX	C	B	S	Memory type and attributes
Flash memory	b000	1	0	0	Normal memory, Non-shareable, write-through
Internal SRAM	b000	1	0	1	Normal memory, Shareable, write-through
External SRAM	b000	1	1	1	Normal memory, Shareable, write-back, write-allocate
Peripherals	b000	0	1	1	Device memory, Shareable

In most microcontroller implementations, the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable. The values given are for typical situations. In special systems, such as multiprocessor designs or designs with a separate DMA engine, the shareability attribute might be important. In these cases refer to the recommendations of the memory device manufacturer.

4 Cache Controller

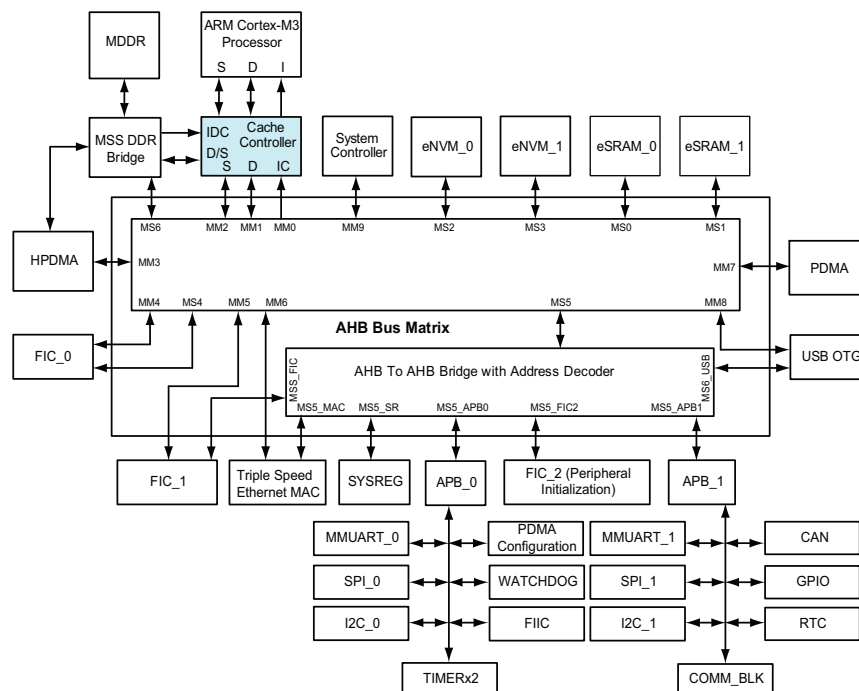
The SmartFusion2 SoC FPGA has an instruction cache. The ARM® Cortex® -M3 processor interfaces to this instruction cache through the Cache Controller. The Cache Controller treats embedded SRAM (eSRAM), embedded nonvolatile memory (eNVM), or DDR memory as main memory.

4.1 Features

- 8 KB of cache size
- Four-way set associativity: Cache Controller has a four-way set associative cache subsystem with 32 byte cache lines organized as 64 sets of 4 cache lines, with a total of 256 locations.
- Cache line size is 32 bytes, fixed irrespective of DDR burst.
- Least recently used (LRU) cache line replacement policy.
- Fill mechanism: Full cache line refill and critical word first.
- The Cortex-M3 processor can write to Cache Memory through the System bus (SBUS).
- Zero wait state in case of a hit (instruction in Cache Memory) and can run up to the maximum system frequency.
- Supports Cache locked mode
- Cache is constructed of latches

The following figure depicts the connectivity of the Cache Controller in a SmartFusion2 device.

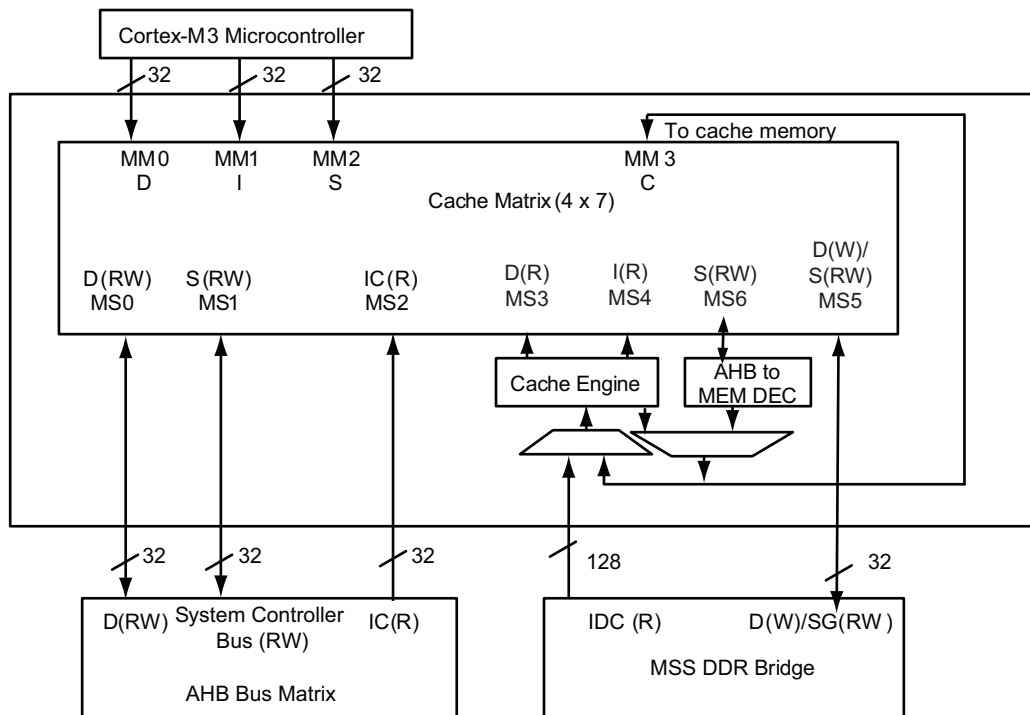
Figure 56 • Cache Controller Interfaces to Cortex-M3 Processor, AHB Bus Matrix, and MDDR Bridge



4.2 Functional Description

The following figure depicts all sub-blocks in the Cache Controller block.

Figure 57 • Cache Controller Block Diagram



MM - Mirrored Master MS - Mirrored Slave DSG - Data System and System Controller Bus
IDC - Icode and DCode Cacheable I - Instruction D - Data S- System R - Read W - Write

The Cache Controller consists of two primary components:

- Cache Matrix
- Cache Engine

4.2.1 Cache Matrix

The cache matrix is a multi-layer AHB-Lite switch matrix. It takes care of the connectivity between masters and slaves, arbitration for slaves, memory mapping between main memory (eNVM, eSRAM, or DDR), and Cache Memory. The masters and slaves in the AHB matrix are referred to as mirrored masters (MM) and mirrored slaves (MS).

One master can access a slave at the same time another master accesses another slave. If more than one master attempts to access the same slave simultaneously, arbitration is performed. Each of the slave devices contains an arbiter, which manages accesses when more than one master attempts to access a slave at the same time.

4.2.2 Memory Mapping

The following sections explain memory mapping for eNVM, eSRAM, and DDR address spaces to cache regions.

4.2.2.1 eNVM Mapping

The cache matrix decodes the code region addresses by any master accessing a targeted slave. By default, the eNVM slave is mapped to the cache.

The following table shows the default memory map of the MSS digital subsystem - eNVM Remapped mode.

Table 88 • Default (eNVM Remapped Mode)

Data/Code Region	Space	Address
CM3 Data Region	Reserved	0xE000_0000 to 0xFFFF_FFFF
	DDR_SPACE 3 (256 MB)	0xD000_0000 to 0xDFFF_FFFF
	DDR_SPACE 2 (256 MB)	0xC000_0000 to 0xCFFF_FFFF
	DDR_SPACE 1 (256 MB)	0xB000_0000 to 0xBFFF_FFFF
	DDR_SPACE 0 (256 MB)	0xA000_0000 to 0xAFFF_FFFF
	eNVM, Remap Area etc (1 GB)	0x6000_0000 to 0x9FFF_FFFF
	Peripheral [SPI, UART, CAN, Fabric etc.] (0.5 GB)	0x4000_0000 to 0x5FFF_FFFF
	Reserved	0x2001_0000 to 0x3FFF_FFFF
	eSRAM-1 (32 KB)	0x2000_8000 to 0x2000_FFFF
	eSRAM-0 (32 KB)	0x2000_0000 to 0x2000_7FFF
CM3 Code Region	Reserved	0x0008_0000 to 0x1FFF_FFFF
	eNVM (Virtual View) [512 KB]	0x0000_0000 to 0x0007_FFFF

The address range of the eNVM_0 is from 0x60000000 to 0x6003FFFF and the address range of eNVM_1 is from 0x60040000 to 0x6007FFFF. The full eNVM (0x60000000 to 0x6007FFFF) is accessible (read/write) in the system space (0x00000000 to 0x0007FFFF). The eNVM AHB controller maps a specified segment of eNVM to this range. This allows multiple firmware images to be stored in eNVM.

Note: Not all devices fully populate either or both eNVM address spaces. Please refer to the SmartFusion2 data sheet for the available eNVM for the device.

4.2.2.2 eSRAM Mapping

The cache matrix supports the ability of re-mapping eSRAM into code space. The two eSRAM blocks are re-mapped to appear at the bottom of the Cortex-M3 processor code space as shown in the following table using eSRAM Remapped mode.

Table 89 • eSRAM Remapped Mode (Memory Map)

Data/Code Region	Space	Address
CM3 Data Region	Reserved	0xE000_0000 to 0xFFFF_FFFF
	DDR_SPACE 3 (256 MB)	0xD000_0000 to 0xDFFF_FFFF
	DDR_SPACE 2 (256 MB)	0xC000_0000 to 0xCFFF_FFFF
	DDR_SPACE 1 (256 MB)	0xB000_0000 to 0xBFFF_FFFF
	DDR_SPACE 0 (256 MB) [MIRRORED]	0xA000_0000 to 0xAFFF_FFFF
	eNVM, Remap Area etc (1 GB)	0x6000_0000 to 0x9FFF_FFFF
	Peripheral [SPI, UART, CAN, Fabric etc.] (0.5 GB)	0x4000_0000 to 0x5FFF_FFFF
	Reserved	0x2001_0000 to 0x3FFF_FFFF
	eSRAM-1 (32 KB) [MIRRORED]	0x2000_8000 to 0x2000_FFFF
	eSRAM-0 (32 KB) [MIRRORED]	0x2000_0000 to 0x2000_7FFF

Table 89 • eSRAM Remapped Mode (Memory Map) (continued)

Data/Code Region	Space	Address
CM3 Code Region	DDR_SPACE 0 (256 MB)	0x1000_0000 to 0x1FFF_FFFF
	Reserved	0x0018_0000 to 0x0FFF_FFFF
	eNVM (Remap View) [512 KB]	0x0010_0000 to 0x0017_FFFF
	Reserved	0x0001_0000 to 0x000F_FFFF
	eSRAM0 & eSRAM1 [64 KB]	0x0000_0000 to 0x0000_FFFF

4.2.2.3 DDR Mapping

In DDR Remapping, the user boot code is present in the DDR. DDR remapping is also used for debugging purposes. This can give high performance execution in systems where DDR is present. The DDR is also used as the main memory for the Cache Controller. In case of DDR remapping, the cacheable region can be configured to 128 MB, 256 MB, or 512 MB. The Cache Controller generates the appropriate DDR address as per remap configuration settings before putting the address on the MDDR bridge.

Table 90 • DDR Remap

Data/Code Region	Space	Address
CM3 Data Region	Reserved	0xE000_0000 to 0xFFFF_FFFF
	DDR_SPACE 3 (256 MB)	0xD000_0000 to 0xDFFF_FFFF
	DDR_SPACE 2 (256 MB)	0xC000_0000 to 0xCFFF_FFFF
	DDR_SPACE 1 (256 MB)	0xB000_0000 to 0xBFFF_FFFF
	DDR_SPACE 0 (256 MB)	0xA000_0000 to 0xAFFF_FFFF
	eNVM, Remap Area etc (1 GB)	0x6000_0000 to 0x9FFF_FFFF
	Peripheral [SPI, UART, CAN, Fabric etc.] (0.5 GB)	0x4000_0000 to 0x5FFF_FFFF
	Reserved	0x2001_0000 to 0x3FFF_FFFF
	eSRAM-1 (32 KB)	0x2000_8000 to 0x2000_FFFF
	eSRAM-0 (32 KB)	0x2000_0000 to 0x2000_7FFF
CM3 Code Region	DDR_SPACE 1 (256 MB)	0x1000_0000 to 0x1FFF_FFFF
	DDR_SPACE 0 (256 MB)	0x0000_0000 to 0x0FFF_FFFF

4.2.3 Memory Maps and Transaction Mapping

The following table depicting transaction mapping depends upon the Memory map mode selected and the possible destination slave for the transaction.

For example, the case eNVM Remapped mode is selected—the condition mentioned in the first row in the table—If the cacheable transaction comes on ICode bus it will be targeted for the eNVM. This transaction initiates on mirrored slave 2 (MS2). The transaction flow will be (MS4 – MM3 – MS2) and it will be routed through AHB Bus Matrix. As shown in the [Figure 57](#), page 134, all the instruction fetch are first checked in the Cache Engine that is MS4 and from there to Cache Memory. If not present, then as shown in the following table, the corresponding routing slave will be selected (For eNVM Remap mode it is switch MS2). The following are the abbreviations used in the table:

IC: Instruction CODE (ICODE) Cacheable

INC: ICODE Non Cacheable

NC: Non Cacheable

DC: Data CODE (DCODE) Cacheable

DNC: DCODE Non Cacheable

(W): Write

(R): Read

Table 91 • Data Path for Various Maps

#	Memory Map Mode	Buses	Supported Trans	Region	Destination Slave	Routed Through
1	Default Memory Map - eNVM Remapped					
		ICODE	IC	eNVM	MS2	AHB Bus Matrix
			INC	eNVM	MS2	AHB Bus Matrix
		DCODE	DC	eNVM	MS2	AHB Bus Matrix
			DNC	eNVM	MS0	AHB Bus Matrix
		System Bus	NC	DDR	MS5	MSS DDR Bridge
			NC	NON DDR	MS1	AHB Bus Matrix
		System Controller Bus	NC	DDR	MS5	MSS DDR Bridge
			NC	NON DDR	MS1	AHB Bus Matrix
2	eSRAM Remapped					
		ICODE	INC	eNVM	MS2	AHB Bus Matrix
			INC	DDR	MS3	MSS DDR Bridge
			INC	eSRAM	MS2	AHB Bus Matrix
		DCODE	DNC	eNVM	MS0	AHB Bus Matrix
			DNC (R)	DDR	MS3	MSS DDR Bridge
			DNC (W)	DDR	MS5	MSS DDR Bridge
			DNC	eSRAM	MS0	AHB Bus Matrix
		SBUS	NC	DDR	MS5	MSS DDR Bridge
			NC	NON DDR	MS1	AHB Bus Matrix
		GBUS	NC	DDR	MS5	MSS DDR Bridge
			NC	NON DDR	MS1	AHB Bus Matrix
3	DDR Remapped					
		ICODE	IC	DDR	MS3	MSS DDR Bridge
			INC	DDR	MS3	MSS DDR Bridge
		DCODE	DC	DDR	MS3	MSS DDR Bridge
			DNC(R)	DDR	MS3	MSS DDR Bridge
			DNC(W)	DDR	MS5	MSS DDR Bridge
		SBUS	NC	DDR	MS5	MSS DDR Bridge
			NC	NON DDR	MS1	AHB Bus Matrix
		GBUS	NC	DDR	MS5	MSS DDR Bridge

Table 91 • Data Path for Various Maps (continued)

#	Memory Map Mode	Buses	Supported Trans	Region	Destination Slave	Routed Through
			NC	NON DDR	MS1	AHB Bus Matrix

4.2.3.1 Unimplemented Address Space

The cache matrix performs address decoding based on the memory map defined, and also to decide which slave is addressed. Any access to RESERVED memory space in code region is considered "unimplemented" from the point of view of the Cache Matrix.

4.2.3.2 Other Features of the Cache Matrix

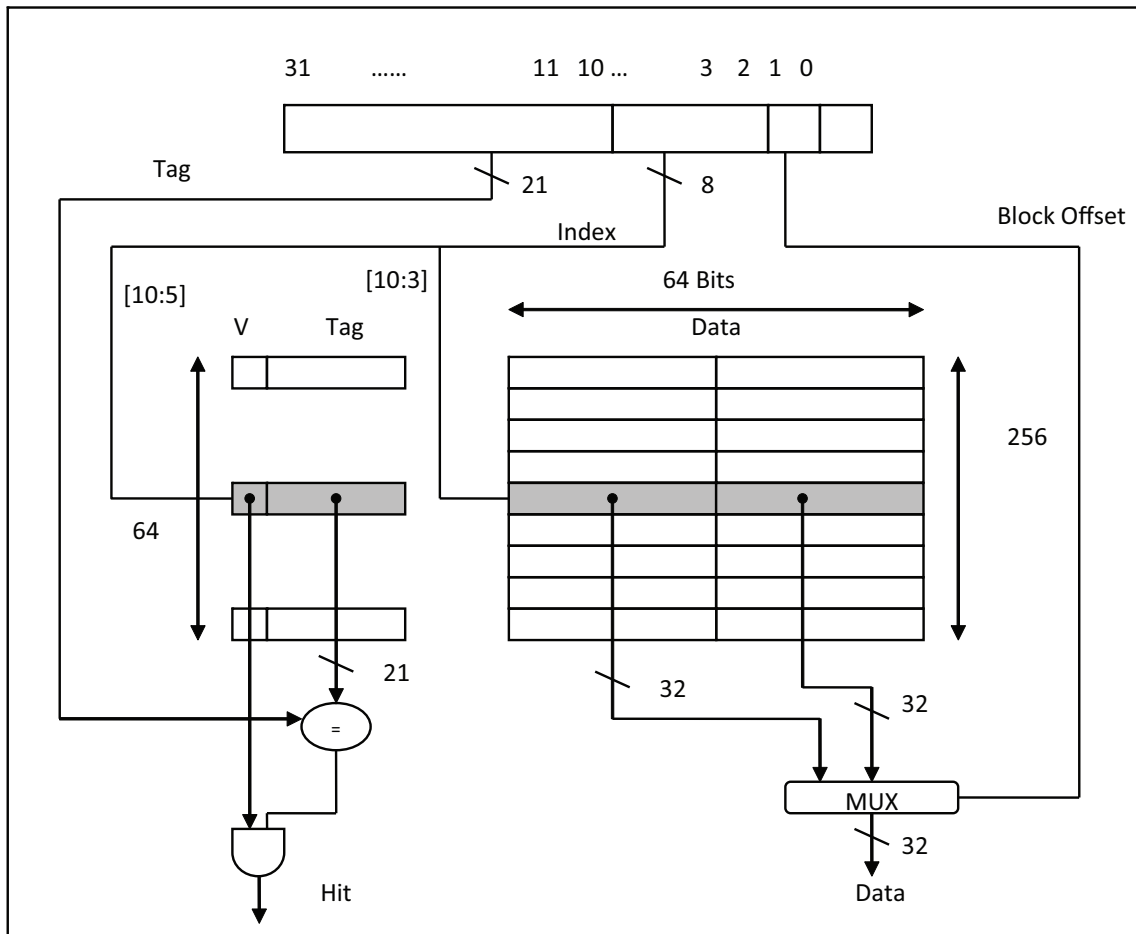
- If any master attempts a write access to an unimplemented address space, the cache matrix completes the handshake with the master, with HRESP error indication. No write occurs to any slave.
- If any master attempts a read access from an unimplemented address space, the cache matrix completes the handshake with the master, with HRESP error indication. Garbage data is returned in this case.
- The cache matrix supports locked transactions from the SBUS towards the eSRAM AHB controller, through the switch, by monitoring HMASTLOCK. The cache matrix initiates IDLE on the AHB bus after every LOCKED transfer. [SmartFusion2 SoC FPGA - Cache Controller Configuration Application Note](#)
- The cache matrix handshakes correctly with masters performing AHB-Lite bursts to any slave. The ICache slave on the cache matrix supports bursts from the cache master.

4.2.3.3 Cache Engine

The Cache Engine takes care of address generation logic using a four-way set associative, hit and miss generation logic, cache line filling/replacement, a temporary local buffer for cache line while writing, and arbitration logic for ICode and DCode buses.

The Cache Controller has a four-way set associative cache subsystem with 32-byte cache lines organized as 64 sets of 4 cache lines. Eight bits from the memory address (shown in the following figure) select one of these 256 different locations. The Cache Controller can map a block of 32 data bytes to any of the cache lines, replacing the LRU block. As one location of the memory contains 64-bit information the required data can be selected by using the second bit from the memory address as shown in the figure.

Figure 58 • General Cache Architecture and Addressing



The Cache Engine has two buses interacting with the ICode and DCode buses through interfaces MS3 and MS4. It supports the following functionalities:

1. Only read transfers from ICode and DCode bus are cached
2. 32 bytes local buffering of cache line read from slave
3. Support 32-/128-bit local interface on the AHB master side
4. All miss non-cacheable transactions targeted for eNVM are routed through MM4
5. Arbitration: In case of simultaneous access from ICode and DCode, all transactions from DCode are processed before ICode is processed.
 - a. Supports full cache flush or index-based flushing

- b. Supports hit/miss generation mechanism for Cache Memory and local buffer
- c. One of the following types of transaction will come to the Cache Engine:
 - Transaction for cacheable region in DDR
 - Transaction for non-cacheable region in DDR
 - Transaction for cacheable region in eNVM
- 6. Supports Cache Disable mode where all transactions will be treated as non-cacheable and replicated “as is” on DDR or switch-side

4.2.3.3.1 Accessing I and D Buses Concurrently

Accessing the I and D buses concurrently is not allowed in the M2S050 devices. In rare cases, accessing the I and D buses concurrently might result in an invalid value returned to the internal registers from the cache causing the firmware to not function properly. All other devices do not exhibit this behavior when using Libero 11.4 SP1 or later.

IAR tool chain users can do a work around for this problem by preventing the Cortex-M3 processor from issuing concurrent I and D buses access through the cache. To implement this work-around, updates are required to the IAR tool chains. All libraries must be fully rebuilt from the source code to avoid this interaction by preventing the cache D-Bus accesses. The user's linker scripts are required to locate constants and data variables outside the memory regions accessed by the cache to prevent conflicts. Consequently, IAR compilation requires using the **-no_literal_pool** option to prevent the compiler/assembler from locating variables close to instructions known as literal pools. Refer to the following two figures. This option prevents literal pool data generation of instructions that used D-bus accesses.

Note: There is no known workaround for SoftConsole, Keil, or GCC (Linux) tool chains.

Figure 59 • IAR Compiler Options

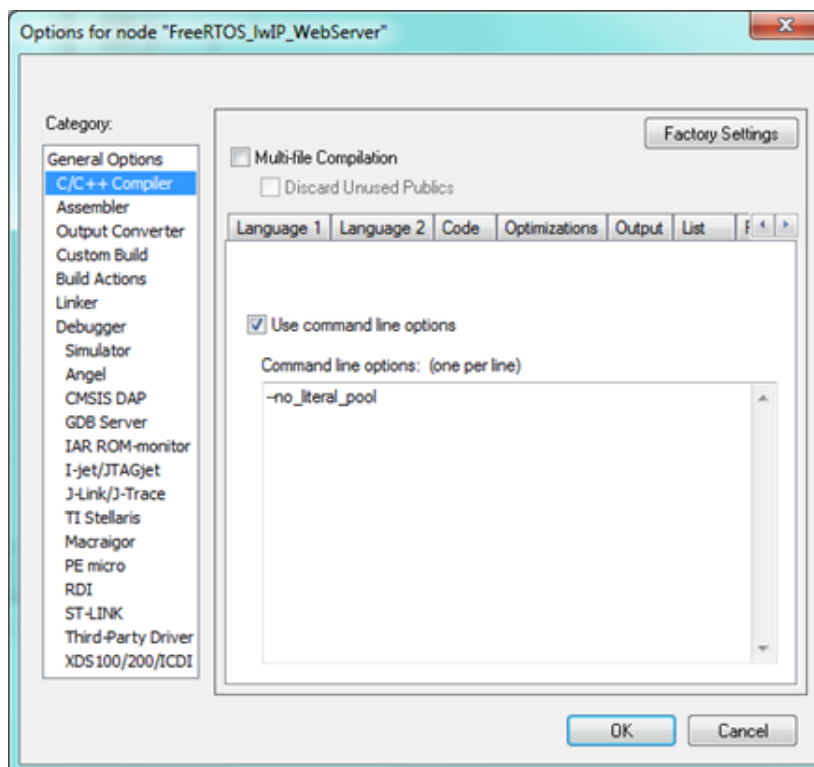
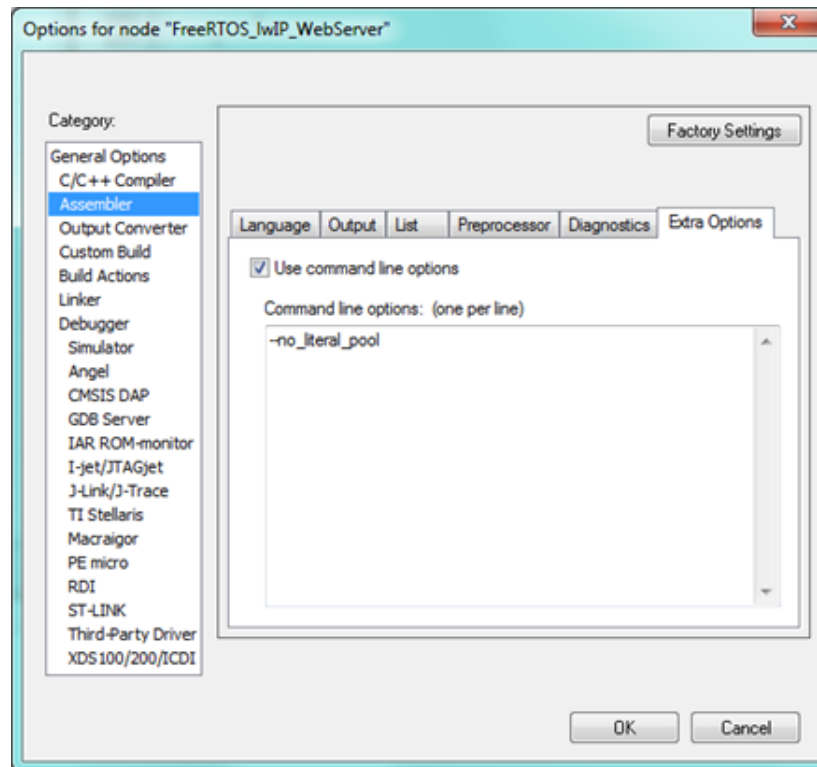


Figure 60 • IAR Assembler Options

4.2.4 Cache Locked Mode

Cache Locked mode is a special mode that provides predictable execution required for some specific applications like avionics and certain security applications. Before enabling Cache Locked mode, the software should ensure that the code is copied to the Cache Memory by simulating a sequential location cache miss through DCode or writes through SBUS by enabling SBUS Write mode. After copying the complete the 8 KB, Cache Locked mode is enabled. After Cache Locked mode is enabled, any access from 0 to 8 KB is directly read from the cache and the cache is not invalidated or refilled for normal operations. The memory region beyond 8 KB is treated as non-cacheable and accessed as per the prevailing memory map.

In Cache Locked mode if an uncorrectable error is detected for cacheable address (0 to 8 KB), then the cache line is fetched from the main memory using the cache lock base address and the entire cache line in Cache Memory is replaced with new data from main memory. Cache Locked mode can only be used in either DDR or eNVM remap modes and the lock base address should be used in the code region of CM3. In Cache Locked mode the least recently used (LRU) cache update algorithm will be deactivated.

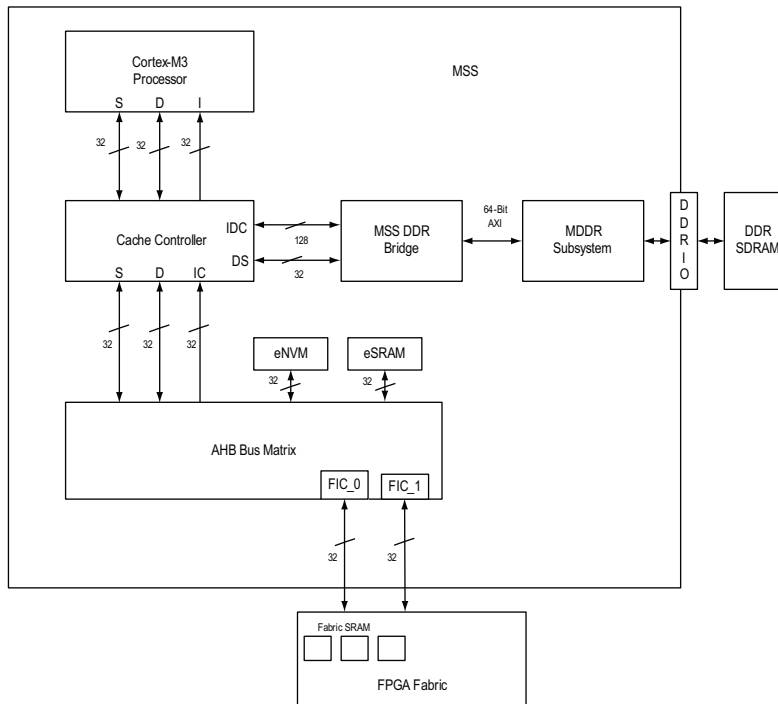
Note: CC_CACHE_LOCK bit of CC_CR system register (Table 660, page 687) is used to lock or unlock the entire 8 KB of Cache memory.

4.2.5 Interfaces

The following figure shows the Cache Controller interface in the MSS subsystem. There are two interfaces through which the Cache Controller is connected to the main memories:

1. Interface towards MDDR bridge: 128-bit AHB-Lite, this interface is read only for instruction/data reads and 32-bit AHB-Lite to access DDR memory through DDR bridge and system bus (read and write access)
2. Interface towards AHB bus matrix: There are three 32-bit AHB-Lite modes:
 - Read/write for non-cacheable data access to eSRAM/eNVM
 - Read/write from SBus
 - Read/write from ICode bus

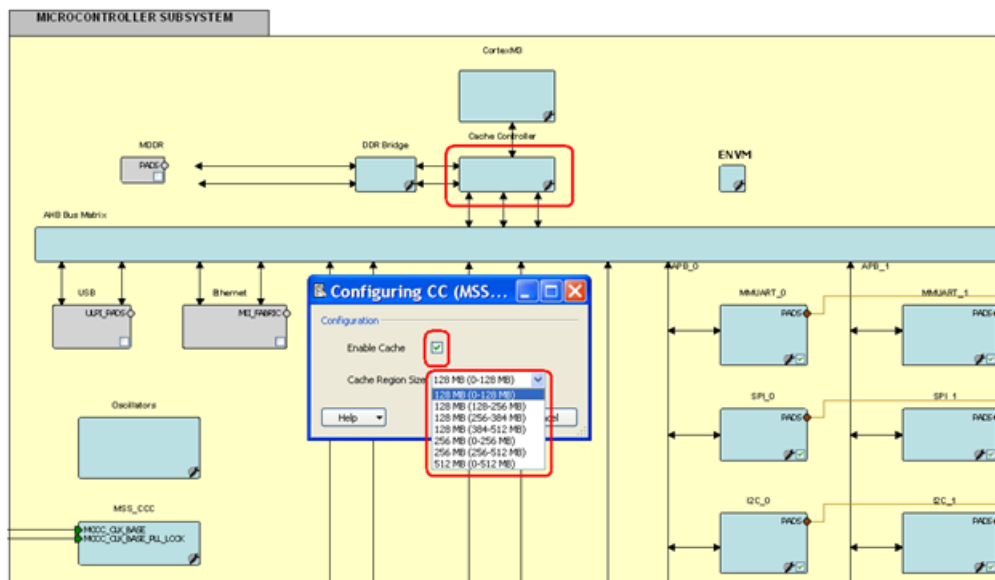
Figure 61 • Cache Controller Interface



4.3 How to Use Cache Controller

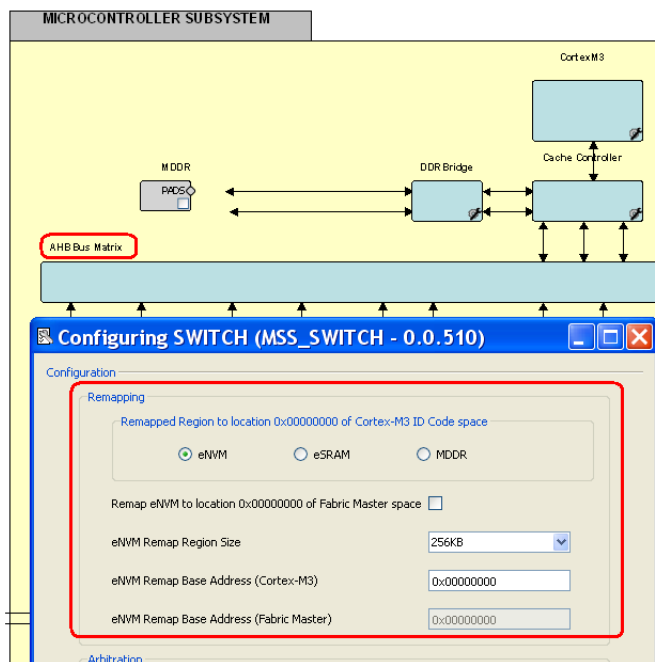
Cache Controller can be configured statically by using the Libero design software. The following figure shows the Cache Controller enable option, cache region size selection.

Figure 62 • MSS Configurator with Cache Controller Configuration Options



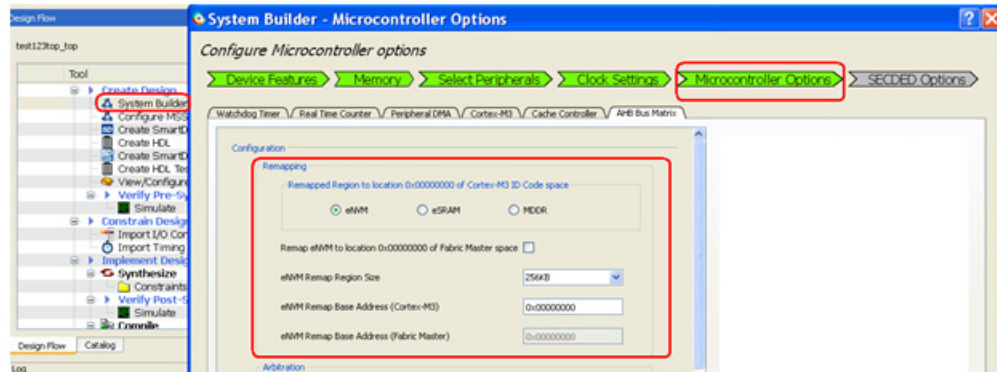
The following figure shows how to select the main memory from memory blocks eNVM, eSRAM, and MDDR.

Figure 63 • MSS Configurator with Remapping Options for eNVM, eSRAM, and MDDR



The selection of the main memory for the Cache Controller can also be made using the system builder flow of the Libero SoC software. This procedure is explained in the following figure.

Figure 64 • System Builder with Remapping Options for eNVM, eSRAM, and MDDR



Cache Controller configurations like enable/disable, selecting the main memory, and Cache Locked mode can also be performed using the firmware/application code with the register settings provided in the [System Registers Used for Cache Operations](#), page 144.

Refer to the following application notes for more details on the Cache Controller configurations:

- [AC389: SmartFusion2 SoC FPGA - Cache Controller Configuration Application Note](#)
- [AC390: SmartFusion2 SoC FPGA – Remapping eNVM, eSRAM, and DDR/SDR SDRAM Memories Application Note](#)

Note: Create or modify the linker scripts/linker settings of the application in such a way that all read and write data sections are in non-cacheable memory regions or accessed through the system bus address space. This note has to be strictly followed if eSRAM or DDR SDRAM are selected as the main memory for the cache.

4.3.1 System Registers Used for Cache Operations

Table 92 • System Registers for Cache Operations

Register Name	Register Type	Flash Write Protect	Reset Source	Description
CC_CR	RW-P	Register	SYSRESET_N	Used to configure cache options like cache enable/disable, cache lock enable/disable, Debug mode system bus read & write.
CC_REGION_CR	RW-P	Register	SYSRESET_N	Defines the cache region size
CC_LOCK_BASE_ADDR_CR	RW-P	Register	SYSRESET_N	If Cache Lock mode is enabled and if there is an ECC error while accessing the Cache Memory, the base address can be used to initiate the transaction to re-read the erroneous data from the system memory which is stored in this register.
CC_FLUSH_INDX_CR	RW-P	Register	SYSRESET_N	Used when Cache Memory index is to be flushed or invalidated.
FLUSH_CR	RW	N/A	SYSRESET_N	Used to flush the Cache Memory

Detailed bit-level descriptions of the cache registers are provided in the [System Register Block](#), page 670.

5 Embedded NVM (eNVM) Controllers

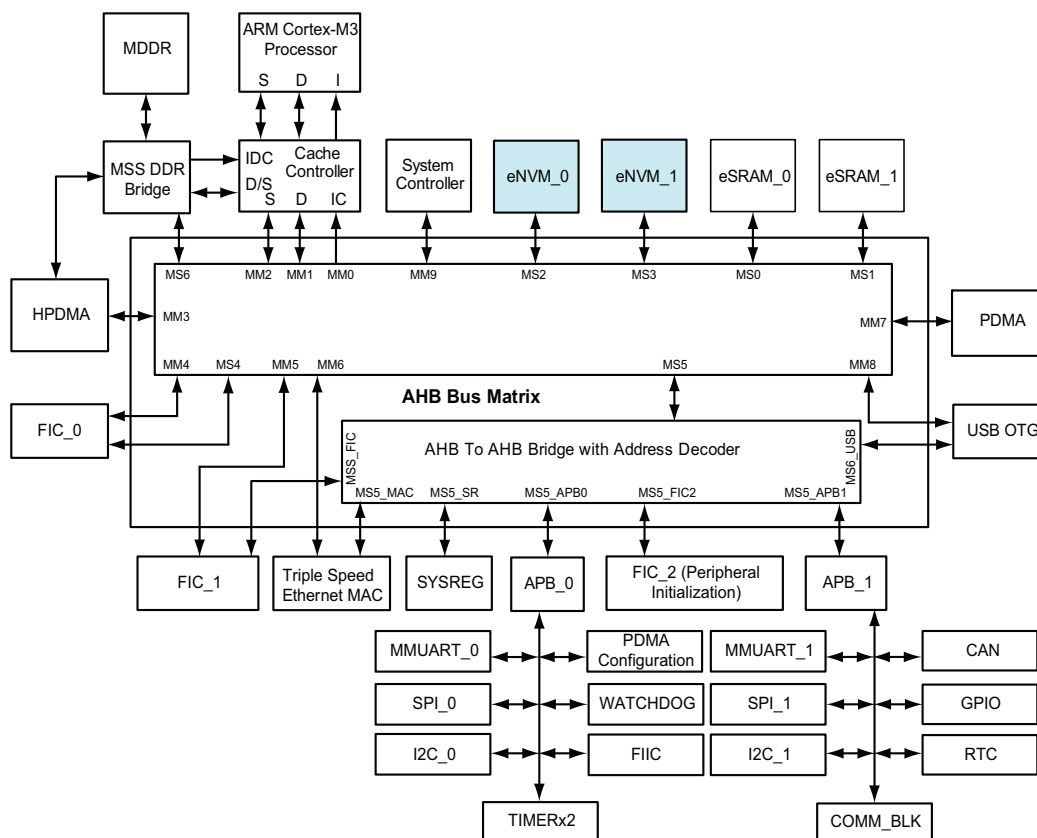
The SmartFusion2 SoC FPGA devices have one or two embedded nonvolatile memory (eNVM) blocks (depending on the device) for user non-volatile memory. The eNVM controller interfaces these eNVM blocks to the advanced high-performance bus (AHB) bus matrix.

5.1 Features

- Single error correction and dual error detection (SECCDED) protected
- Based on the selected SmartFusion2 device, the total size of eNVM memory ranges from 128 KB, 256 KB, and 512 KB.
 - M2S005 has a single block of 128 KB.
 - M2S010, M2S025, M2S050, and M2S060 have a single block of 256 KB.
 - M2S090 and M2S150 have two blocks of 256 KB each. The total eNVM size is 512 KB.
- In devices with two blocks present, any two masters can access the eNVM blocks (eNVM_0 and eNVM_1) in parallel, which improves the overall performance of the system.

As shown in the following figure, the eNVM block(s) is connected as slave to the AHB bus matrix.

Figure 65 • eNVM Connection to AHB Bus Matrix



5.2 Functional Description

The address range of eNVM_0 is 0x60000000 to 0x6003FFFF and the address range of eNVM_1 is 0x60040000 to 0x6007FFFF. The location of eNVM_1 always follows eNVM_0 in the system memory map. The following table gives the eNVM_0 and eNVM_1 addresses for different devices.

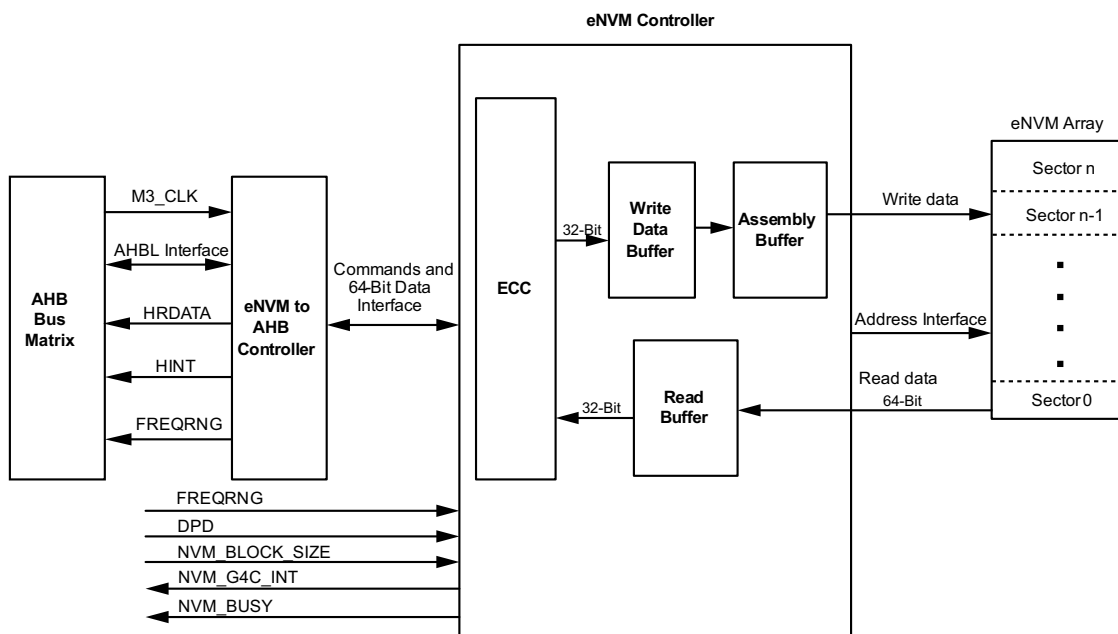
Table 93 • eNVM Address Locations

Device	eNVM_0	eNVM_1	Total NVM
M2S005	0x60000000	None	128 KB
M2S010	0x60000000	None	256 KB
M2S025	0x60000000	None	256 KB
M2S050	0x60000000	None	256 KB
M2S060	0x60000000	None	256 KB
M2S090	0x60000000	0x60040000	512 KB
M2S150	0x60000000	0x60040000	512 KB

Both eNVMs and embedded NVM controllers are identical and the eNVM controller consists of three components:

- eNVM Array
- eNVM Controller
- eNVM to AHB Controller

Figure 66 • eNVM Controller Block Diagram



M3_CLK is used within the MSS to clock the AHB bus matrix. Refer to [UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide](#) for more information on M3_CLK.

eNVM Array: The eNVM array is connected to a 25 MHz internal oscillator. This 25 MHz internal oscillator is used during device start up to initialize the NVM controller. It is also used for eNVM program operation. For other eNVM operations (Read and Verify), the eNVM controller operates at the M3_CLK. During eNVM read operations, the NVM controller uses the NV_FREQRNG input to insert wait states to match with the eNVM array access times. The eNVM array stores the data. [Table 94](#), page 147 shows the eNVM memory organization and the total size of the eNVM.

eNVM Controller: Decodes all transactions from the AHBL master and issues the commands to the eNVM array.

ECC: The error-correcting code (ECC) block in eNVM Controller performs the SECDED. The ECC stores error correction information with each block to perform SECDED on each 64-bit data word. ECC does not consume any eNVM array bits. Refer to [Table 113](#), page 184 for ECC status information. ECC block in eNVM Controller is enabled by default. The user has no access to control the ECC block.

Read Data Buffer: Contains four 64-bit data words. It functions as a small cache by reading NVM data as four consecutive 64-bit data words. Data read from the eNVM is stored in read data buffer (RDBUFF) and presented to AHB read data bus (HRDATA) corresponding to HADDR.

If the data is not available, an eNVM read cycle is invoked to retrieve data from the eNVM array. To support an 8-bit fixed length wrapping burst, four eNVM read cycles are automatically invoked and data read from the eNVM is stored in RDBUFF. Read data is presented to HRDATA when the data for the current read address becomes available.

Assembly Buffer (AB): The eNVM is page-based flash memory. Only one page of data (1,024 bits) can be written at a time. The assembly buffer stores thirty-two 32-bit data words for programming. During programming, the assembly buffer cannot be updated. If more than one page is to be written, the page programming function needs to be called as many times as the number of pages.

Write Data Buffer: The write data buffer provides a secondary 32-word data buffer. This can be updated with the next 32 words to be programmed during eNVM programming.

eNVM to AHB Controller: This block interfaces the eNVM Controller with the AHB-Lite (AHBL) master as shown in [Figure 66](#), page 146.

5.2.1 Memory Organization

The eNVM is divided into sectors based on the eNVM size. Each sector is divided into 32 pages. Each page holds 1,024 bits of data. The following table lists the total available memory and its organization.

Table 94 • Memory Organization

Device	NVM Size	Number of Sectors	Pages per Sector	Bytes per Page	Words per Page	64-Bit Locations per Page	Total Bytes
M2S005	128 KB	32	32	128	32	16	131072
M2S010	256 KB	64	32	128	32	16	262, 144
M2S025	256 KB	64	32	128	32	16	262, 144
M2S050	256 KB	64	32	128	32	16	262, 144
M2S060	256 KB	64	32	128	32	16	262, 144
M2S090	512 KB (two eNVMs, each 256 KB)	64 per NVM	32 per NVM per sector	128	32	16	262, 144 per NVM
M2S150	512 KB (two eNVMs, each 256 KB)	64 per NVM	32 per NVM per sector	128	32	16	262, 144 per NVM

5.2.2 Data Retention Time

The following table shows the retention time of the eNVM with respect to the number of programming cycles. The same values are applicable for both commercial and industrial SmartFusion2 product grades. Refer to [DS0128: IGLOO2 FPGA and SmartFusion2 SoC FPGA Datasheet](#) for more information on Programming cycles and retention time.

Table 95 • Data Retention Time

Programming Cycles Per eNVM Page	Retention
< 1000	20 years
< 10000	10 years

Note: The eNVM is not prevented from programming, even if a page exceeds the write count threshold. The eNVM Controller generates a flag through Status register.

5.2.3 eNVM Access Time

Refer to the **Embedded NVM (eNVM) Characteristics** section from [DS0128: IGLOO2 FPGA and SmartFusion2 SoC FPGA Datasheet](#) for eNVM Maximum Read Frequency and eNVM Page Programming Time.

5.2.4 Theory of Operation

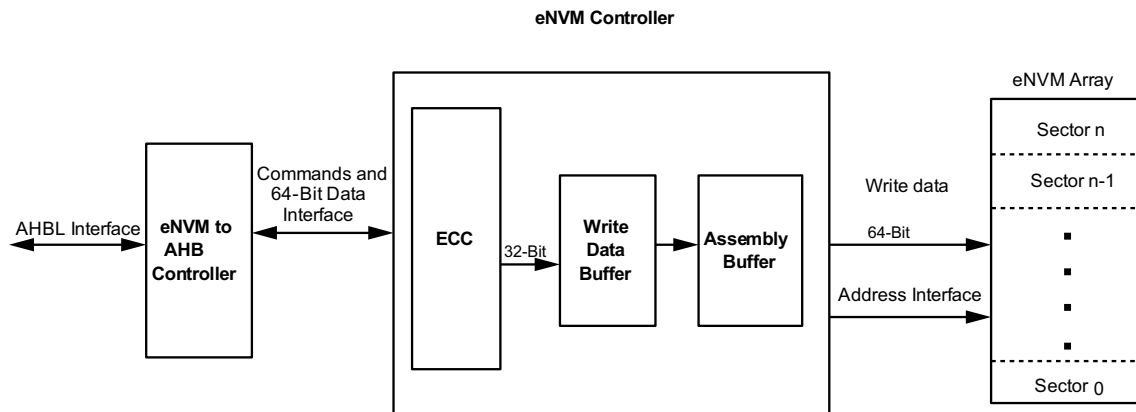
The eNVM AHB Controller supports the following operations:

- Interface from AHBL for read, write, and erase operations
- Issues all eNVM commands through AHBL read and write bus operation. The data width to and from AHBL bus is 32 bits, and data to and from eNVM is 64 bits.
- AB can be read directly from AHBL bus.
- eNVMs treated as ROM. AHBL write transactions to eNVM user data array receive errors on HRESP and write will be ignored.
- Page Program command is used to write the NVM user data array.
- AB can be written directly or loaded from the write data buffer (WDBUFF). Data can be written to WDBUFF in byte, half-word or word AHB transfers.
- Data for Page Program comes from WDBUFF or user data previously written into AB.
- Command code in [Table 98](#), page 151 determines the NVM commands to be issued. The eNVM user data array is treated as ROM, so any program operations must be performed by submitting relevant commands to the controller. Any AHBL writes to NVM user data without a valid NVM command will cause the HRESP signal to be asserted on the AHBL bus. Any data that needs to be written into the NVM user array must be uploaded first to the WDBUFF and then written into the NVM user array through the assembly buffer. Program operation for the NVM user array occurs at the page boundaries.

5.2.4.1 Write Control

The data to be programmed into eNVM must first be uploaded into WDBUFF due to the width difference between the AHBL bus and the eNVM. Data can be written into WDBUFF by word, half-word, or byte from the AHBL bus. ProgramDa and ProgramADS commands take care of uploading data into AB from WDBUFF before programming eNVM.

Data is sent to eNVM from WDBUFF in chunks of double words (64 bits). Subsequent data transfer commands to the AB and then to eNVM array, or commands such as ProgramAd, ProgramDa, and ProgramStart, must specify the page address and upload data to AB to start eNVM array programming. Refer to [Table 98](#), page 151 for more information on commands.

Figure 67 • Write Path

5.2.4.2 Read Control

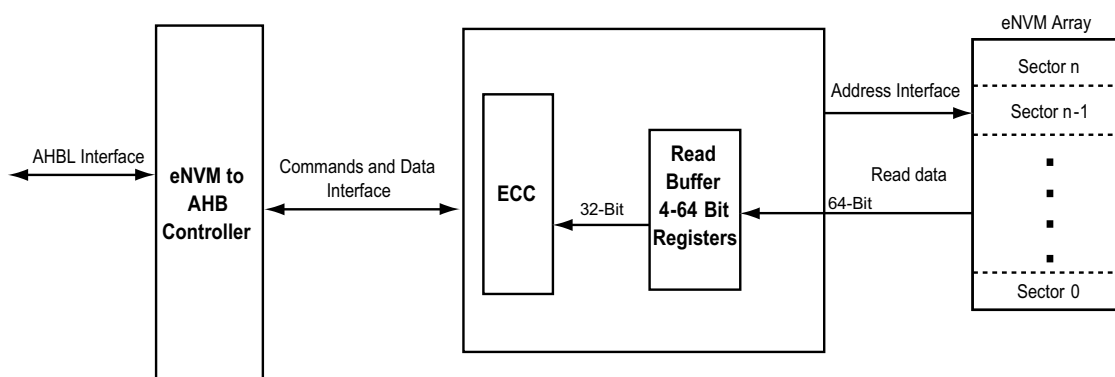
The following steps describe eNVM read control.

- The read transaction from the eNVM user array to AHBL bus uses the read data buffer as a mini cache.
- If the requested 32-bit word exists in the read data buffer, it will be returned immediately on the AHB bus; otherwise a 64-bit read access of the eNVM is initiated and will take several clock cycles as configured by [ENV_M_CR](#) register.
- The eNVM data is stored in the read data buffer and provided to the AHB bus. Assuming that the eNVM address is incremented, the data value stored in the read data buffer is available for the next AHB read cycle.

The following figure shows the eNVM array read path.

The AHB Controller also supports WRAP4 burst operations, which are initiated by the cache controller. In this case, the AHB eNVM controller will automatically perform four 64-bit read operations (critical word first) and fill the read data buffer in advance to the AHB read transactions to increase system throughput.

Figure 68 • Read Path



In the eNVM array, the addresses are 64-bit locations; therefore each page of 1,024 bits (16 double words = 32 words) requires an AHBL address map, as specified in the following table.

Table 96 • AHBL Address Map to NVM

Sector Number	Page Number in Sector	Address in Page	Byte Number in 64-Bit Data
HADDR[17:12]	HADDR[11:7]	HADDR[6:3]	HADDR[2:0]

When programming the eNVM, sector and page addresses must be programmed into the command (CMD) register, as specified in [Table 98](#), page 151.

5.2.4.3 eNVM Commands

The eNVM commands are explained in the [Table 98](#), page 151. The eNVM Command register is used to program the eNVM commands. The following section explains the details of the eNVM Command register.

5.2.5 eNVM Command Register

The following table shows the Command register bit definitions.

Table 97 • Command (CMD) Register

Bit	Description
31:24	Command code
23:0	Address field; to supply address for NVM operation, refer to Table 98 , page 151.

The Command register is located at offset 0x148 in the Control register. Refer to [Table 98](#), page 151 for more information. By writing to CMD when HADDR[18:0] = 0x148, any eNVM operation may be invoked. The eNVM goes into a busy state and HREADY is set High until it finishes the write operation. Any further invoking of the eNVM operation will cause HREADY to go Low until it finishes the previous operation.

The following steps describe when to write to the Command Register, decoding of commands and command execution.

- The command register should only be written when the NVM is non-busy (Status Register bit 0). Refer to [Table 113](#), page 184 for the Status Register definitions
- If the Command register is written when the NVM is still busy from a previous command then the logic will prevent the new command and all future commands, the access_denied bit in the STATUS register will be set. To recover from this state, 1 should be written to bit 1 in the [CLRHint\[2:0\]](#) register to clear the access_denied bit. This mechanism is used to detect the improper NVM command sequences and protect the NVM data until the firmware recovers.
- When the AHBL triggers a write transaction with HADDR[18:0] = 0x148, HWDATA is treated as a command (CMD).
- CMD[31:24] will be decoded as the eNVM operation, as mentioned in [Figure 68](#), page 150.
- The value from CMD[23:3] will be decoded as the NVM array address for the eNVM operation. Depending on the command code, some LSB bits of CMD[23:0] will be ignored. For example, to submit a program address, only the page address CMD[17:7] is significant. Therefore CMD[17:7] is taken as the NVM address and CMD[6:0] is ignored. Refer to [Table 98](#), page 151 for more information.

For masters, which are only capable of byte access, four cycles of write may be needed to fill the Command (CMD) register, by writing to 0x14b, 0x14a, 0x149, and 0x148.

Table 98 • Command Table

Name	HADDR		HWDATA		Transaction Type	Description
	18	17:0	31:24	23:0		
Read Page	0	AA	X	X	Read	
ProgramAd	1	ACMD	05	PGA	Write	Submit page address for programming. CMD[17:7] is considered as the eNVM address and CMD[6:0] is ignored.
ProgramDa	1	ACMD	06	AAB	Write	Submit data to assembly buffer for programming, up to 16 dwords can be written to the assembly buffer as specified by DWSIZE. ProgramDa must be preceded by ProgramAd. CMD[17:7] is considered as the eNVM address and CMD[6:0] is ignored.
ProgramStart	1	ACMD	07	X	Write	Start program NVM operation

Table 98 • Command Table (continued)

Name	HADDR		HWDATA		Transaction Type	Description
	18	17:0	31:24	23:0		
ProgramADS	1	ACMD	08	PGA	Write	Start whole program page procedure, includes sending page address, sending entire content of write data buffer to assembly buffer, then starting the NVM operation.
VerifyAd	1	ACMD	0D	PGA	Write	Submit page address for standalone verify. CMD[17:7] is taken as the eNVM address and CMD[6:0] is ignored.
VerifyDa	1	ACMD	0E	AAB	Write	Submit data to assembly buffer for standalone verify. Up to 16 dwords can be written to the assembly buffer, as specified by DWSIZE. VerifyDa must be preceded by the VerifyAd. CMD[6:3] is taken as the starting double word address and CMD[23:7] is ignored.
VerifyStart	1	ACMD	0F	X	Write	Start standalone verify NVM operation
VerifyADS	1	ACMD	10	PGA	Write	Start whole standalone verify procedure; includes sending page address, sending entire content of write data buffer to assembly buffer, and then starting NVM operation.
User Unlock			13	X	Write	Submit a User Unlock NVM command before Program NVM.

Note:

- AA = NVM Array address. Refer to [Table 93](#), page 146.
- AAB = Address of assembly buffer. Refer to [Table 112](#), page 180 for address values.
- ACMD = Address of CMD register. The Command register is located at offset 0x148 in the Control register. Refer to [Table 112](#), page 180 for more information.
- PGA = Page address
- SEA = Sector address
- X = Not used

5.2.5.1 Read Page

Data read from eNVM is stored in the read data buffer (eight 32-bit memory blocks) and presented to HRDATA based on HADDR[2:0]. For non-sequential reads, the read data buffer is checked first. If the data is available, it is presented to HRDATA; otherwise an eNVM read cycle is invoked to read the data from the eNVM array and data is presented to HRDATA as soon as corresponding data is available.

To support 8-byte fixed length burst (that is, to read the complete read data buffer, which consists of eight 32-bit memory blocks), 4 eNVM read cycles (each 64-bit) are automatically invoked. Data read from the eNVM is stored in the read data buffer.

5.2.5.2 Page Program

This mode allows writing the page with pre-erase. In Page Program there are three stages:

- **ProgramAd:** This command is used to submit the page address to be programmed.
- **ProgramDa:** Once the ProgramAd command is issued, data can be written to AB.
- **ProgramStart:** After ProgramAd and ProgramDa (optional), ProgramStart can be used to start the NVM operation. Once the NVM operation starts and until it finishes, any further NVM accessing AHBL transaction will result in HREADYOUT going Low until the operation is done.

If the command ProgramDa is not issued after the ProgramAd operation, the current data in the assembly buffer will be programmed to the NVM array.

5.2.5.2.1 Program Page with a Single AHBL Write

ProgramADS: During the command ProgramADS, a single AHBL write transaction can be used to start and complete the program page procedure. By default, all WDBUFF content is written to AB and internal program operation automatically begins. Once the NVM operation starts and until it finishes, any further NVM accessing AHBL transaction will result in HREADYOUT going Low until the operation is done.

Note: eNVM frequency range (NV_FREQRNG field of [ENVM_CR](#) system register) value must be set to maximum value 15 to ensure the correct programming of the eNVM. After programming eNVM, restore the original frequency range value for eNVM read or verify operations.

5.2.5.3 Standalone Verify

This mode allows verifying the operation of a page. In verify there are three stages:

- **VerifyAd:** This command is used to submit the page address to be verified.
- **VerifyDa:** Once the VerifyAd command is issued, data can be written to AB.
- **VerifyStart:** After VerifyAd and VerifyDa (optional), VerifyStart can be used to start the NVM operation. Once the NVM operation starts and until it finishes, any further NVM accessing AHBL transaction will result in HREADYOUT going Low until the operation is done. If the VerifyDa command is not issued after the VerifyAd operation, the current data in assembly buffer is verified with the NVM array.

5.2.5.3.1 Standalone-Verify with a Single AHBL Write

VerifyADS: With the command VerifyADS, a single AHBL write transaction can be used to start and complete the verify page procedure. By default, all WDBUFF content is written to AB and the internal Standalone-Verify operation automatically starts. Once the NVM operation starts and until it finishes, any further NVM accessing AHBL transaction will result in HREADYOUT going Low until the operation is done.

5.2.5.4 Set Lock Bit and User Unlock Commands

There is a user page lock bit to lock the page for writing. The Control Register PAGE_LOCK_SET[0] is used to set the user lock bit of the page. Refer to PAGE_LOCK_SET register in [Table 112](#), page 180 for more information. If PAGE_LOCK_SET[0] == 1, then nv_s_page_lock_set will be asserted when submitting the address for Program.

To program a page, the User Unlock command must be submitted before submitting ProgramAd or ProgramADS.

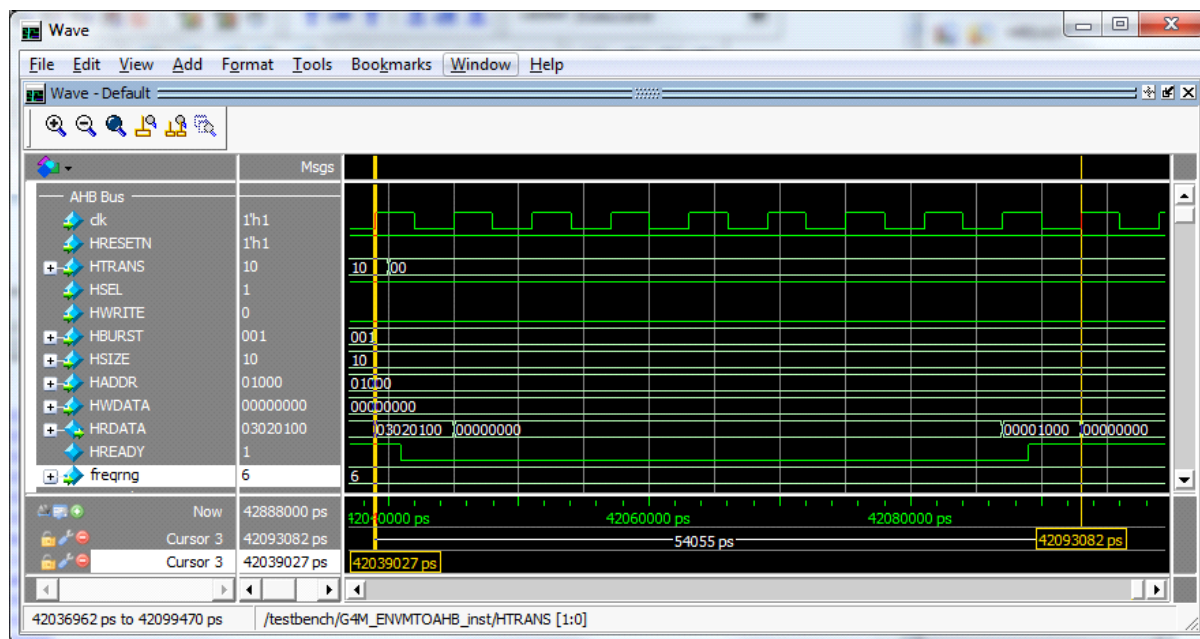
5.2.5.5 eNVM Read Operations with Timing Diagrams

The following are the example eNVM read operations with the Cortex-M3 processor operating at 166 MHz. The eNVM NV_FREQRNG is set to 6.

5.2.5.5.1 Single Word Read

The following figure shows the AHB read command to 0x60001000 starting at the first cursor, and data being returned at the second cursor 9 clock cycles later.

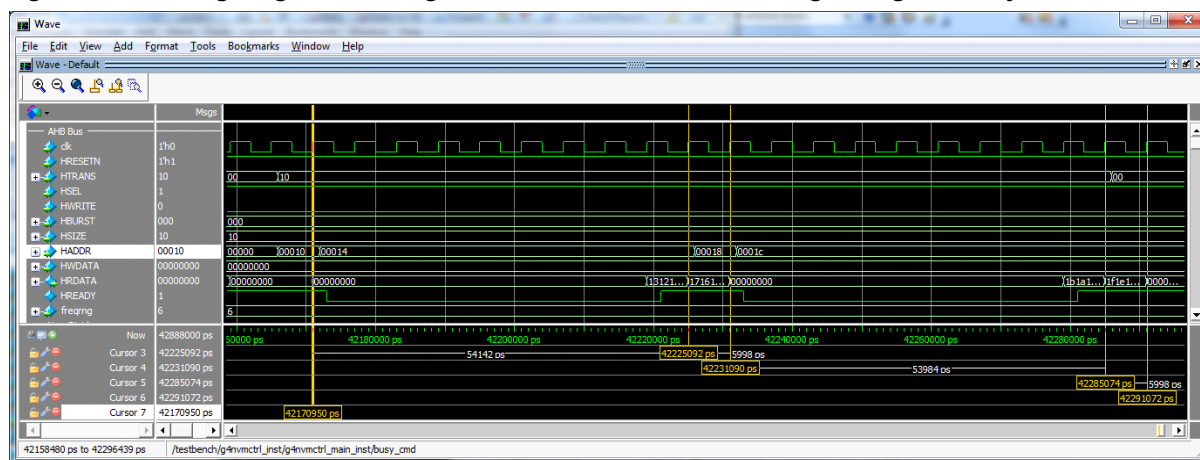
Figure 69 • Timing Diagram Showing Single Word Read Operation



5.2.5.5.2 Consecutive Reads Incrementing through Memory

In this case, four reads from addresses 0x60000010, 0x60000014, 0x60000018, and 0x6000001C are initiated by the AHB master in succession. The first word is returned 9 clock cycles later (as shown in the preceding figure), but the second word occurs in the following cycle, 9 clock cycles later the third word is provided and the fourth word occurs in the next clock cycle. This pattern is repeated as the memory is incremented as shown in the following figure.

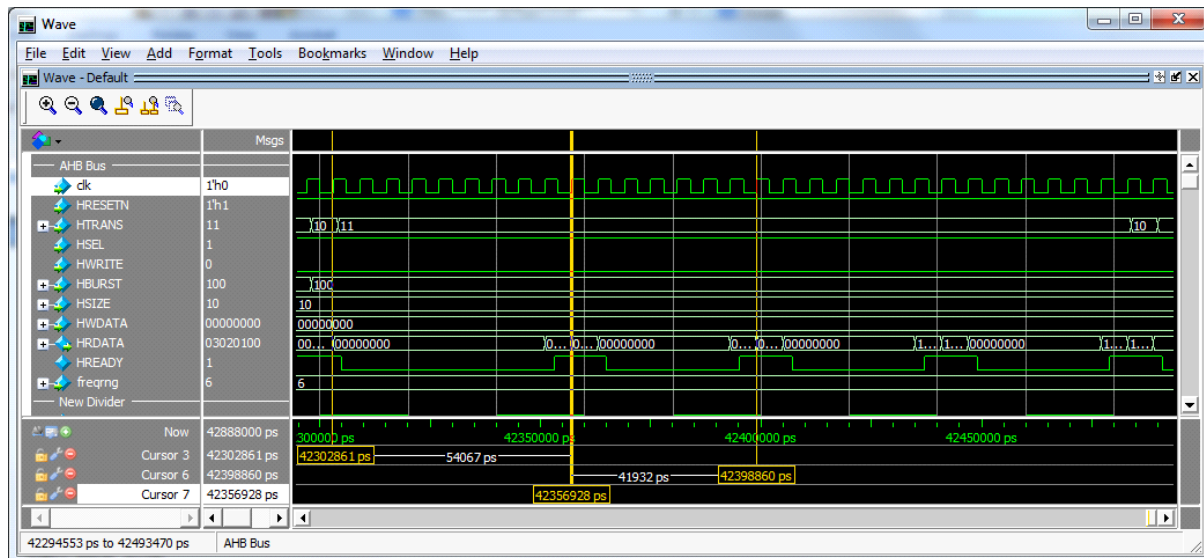
Figure 70 • Timing Diagram Showing Consecutive Reads Incrementing through Memory



5.2.5.5.3 Cache Fill Operation Utilizing Bursts

The internal cache fill operations using AHB wrapping can utilize bursts to optimize the cache fill operations. The AHB-NVM controller always returns 8 words in a burst. The first word returns after 9 clock cycles, and second word in the following cycle as shown in the preceding figure. But the third word occurs 7 clock cycles later, and the fourth word occurs a cycle later with a repeating pattern for the remaining words as shown in following figure. This burst transfer is 8 clock cycles quicker than a non-burst sequence of read commands.

Figure 71 • Timing Diagram Showing Cache Fill Read Operations Utilizing Bursts



5.2.5.6 eNVM Program and Verify Operations Timing Diagrams

Timing diagrams in this section illustrate eNVM Program and Verify operations at the AHB bus transfer level with the Cortex-M3 processor operating at 166 MHz. The eNVM NV_FREQRNG is set to 15. The sample eNVM operation programs the eNVM sector 0 page 4 with random data and verifies the eNVM sector 0 page 4.

Note: In all the waveforms, the eNVM controller register offset is shown in AHB address line (HADDR). Refer to [eNVM Control Registers](#), page 180 for more information.

5.2.5.6.1 Sequence of eNVM Program and Verify Operations when using ProgramADS and VerifyADS Commands

The following figure shows the following sequence of eNVM ProgramADS and VerifyADS commands:

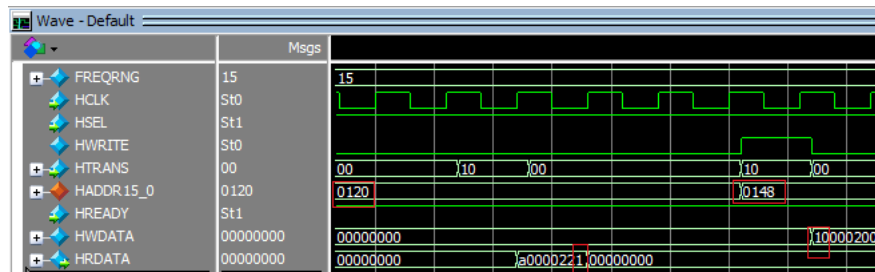
1. Cortex-M3 master requests for exclusive register access by writing 0x1 to the REQACCESS register.
2. Fills the WDBUFF (Write Data Buffer) register with the data to be written to the eNVM array.
3. Issues ProgramADS command.
4. Completes the eNVM Program operation and starts the eNVM Verification by issuing a VerifyADS command.
5. Completes the eNVM verify operation.
6. Releases the exclusive register access by writing 0x0 to the REQACCESS register.

The status of the eNVM operations are monitored by polling the Status register response.

For a description of the registers, see [Table 112](#), page 180.

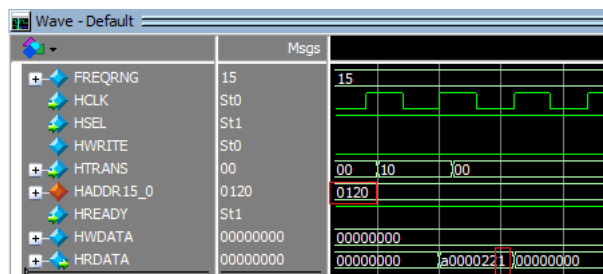
The following figure shows completion of ProgramADS and issue of VerifyADS command.

Figure 75 • Completion of ProgramADS and Issue of VerifyADS Command



The ProgramADS command completion can be confirmed by polling Status register response. The following figure shows the completion of eNVM verify operation.

Figure 76 • Completion of eNVM Verify Operation



5.2.5.6.2 Sequence of eNVM Program and Verify Operations when Using ProgramAD, ProgramDA, ProgramStart, VerifyAD, VerifyDA, and VerifyStart Commands

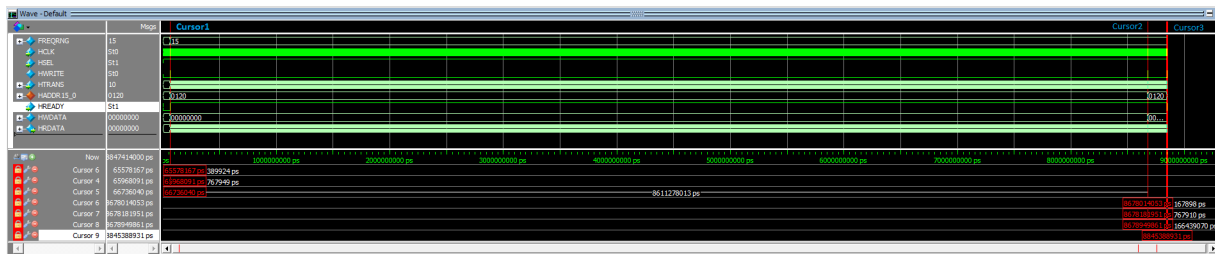
Figure 77, page 158 through Figure 81, page 159 show the sequence of eNVM program operation:

1. Cortex-M3 master requests for exclusive register access by writing 0x1 to the REQACCESS register. Refer to Figure 78, page 158.
2. Fills the WDBUFF (Write Data Buffer) register with the data to be written to the eNVM array. Refer to Figure 78, page 158.
3. Issues ProgramAD command. Refer to Figure 79, page 158.
4. Completes the ProgramAD command and Issues the ProgramDA command. Refer to Figure 80, page 158.
5. Completes the ProgramDA command and Issues ProgramStart command. Refer to Figure 81, page 159.
6. Completes the eNVM Program operation and starts the eNVM verification by issuing a VerifyAD command.
7. Completes the VerifyAD command and Issues the VerifyDA command.
8. Completes the VerifyDA command and Issue the VerifyStart command.
9. Completes the eNVM verify operation.
10. Releases the exclusive register access by writing 0x0 to the REQACCESS register.

The status of the eNVM operations are monitored by polling the Status register response.

The following figure shows the complete eNVM program and eNVM verify operations.

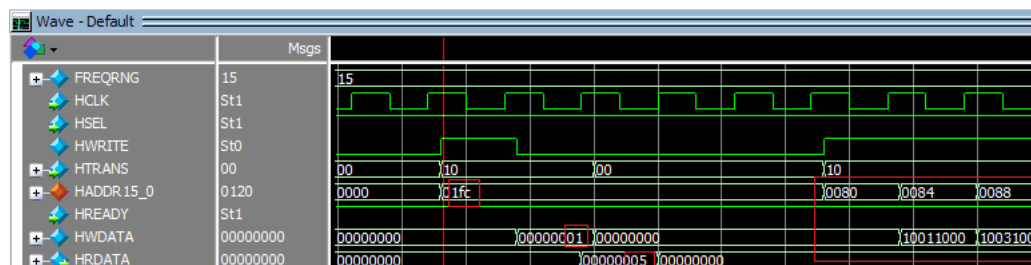
Figure 77 • Complete eNVM Program and Verify Operations Waveform



At cursor 1, steps 1 and 2 in the sequence are performed. At cursor 2, the eNVM ProgramStart operation is completed and VerifyAD operation is started. At cursor 3, the verify operation is completed. Refer to the preceding figure. The eNVM commands sequence is explained in waveforms in [Figure 78](#), page 158 through [Figure 81](#), page 159.

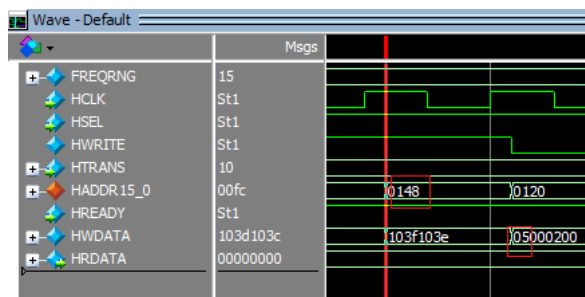
The following figure shows the Cortex-M3 master requesting for exclusive register access and filling WDBUFF (Write Data Buffer).

Figure 78 • Exclusive Register Access and Filling Data in WDBUFF



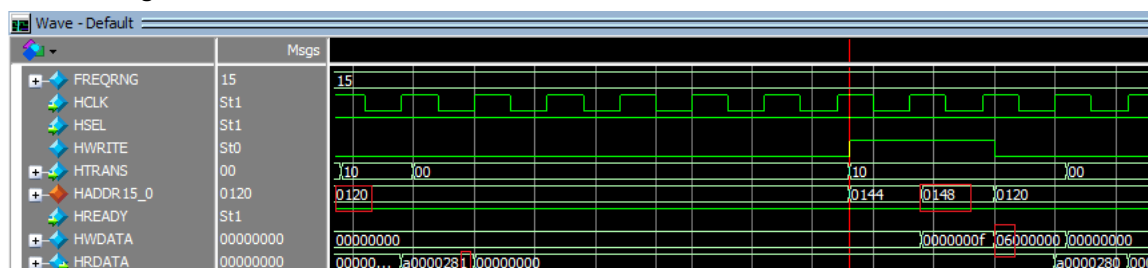
The following figure shows the issuance of the ProgramAD command.

Figure 79 • ProgramAD Command



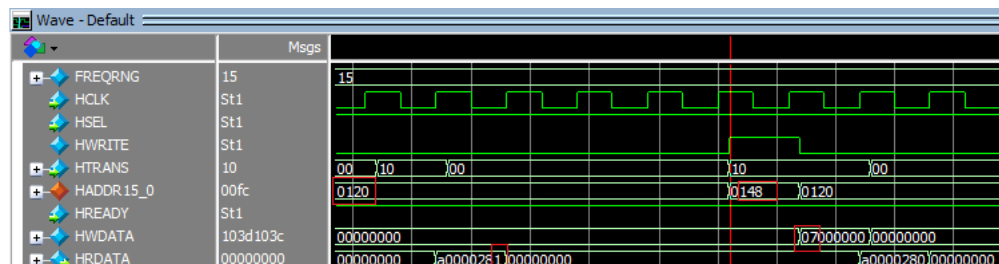
The following figure shows the completion of ProgramAD command and the issuance of the ProgramDA command.

Figure 80 • ProgramDA Command



The following figure shows the completion of the ProgramDA command and the issuance of the ProgramStart command.

Figure 81 • ProgramStart Command



The completion of the eNVM command is confirmed by monitoring the eNVM status register for eNVM ready and the next command in sequence is sent. VerifyAD, VerifyDA, and VerifyStart commands are issued by writing corresponding command value into CMD register.

5.2.6 Error Response

The error response, which is indicated by the HRESP signal, is asserted if any of the following conditions occur:

- AHBL burst read is terminated early or address sequence is not as expected. This should never occur within the system during normal operation.
- AHBL write transaction addressed to read-only user data array
- AHBL read or write transaction to a protected memory area. Refer to [Security](#), page 159.

Data on HRDATA with error response is zero. A write transaction addressed to read-only Control register such as RD or RDT will not trigger an error response. However, the data in these registers will not be affected.

5.2.7 Interrupt to Cortex-M3 Processor

Setting the Control registers `INTEN[10:0]`, as shown in [Table 112](#), page 180, allows the user to configure HINT (INTISR[17] and INTISR[18] of Cortex-M3 processor) to assert an interrupt on any active status events from eNVM, such as the assertion of any status bit from eNVM or when an internal eNVM operation ends.

After HINT is asserted, the Cortex-M3 processor determines the next steps. The Cortex-M3 processor can respond to the interrupt and then clear HINT by writing 1 to bit 0 of the write-only register `CLRHint[2:0]` (HADDR = 0x158) in [Table 112](#), page 180. If the Cortex-M3 processor decides to ignore the interrupt (by masking it out), the interrupt is cleared if read or write continues and the interrupt-triggering events are not re-occurring. If the same triggering event happens again, HINT will remain asserted.

5.3 Security

The eNVM is protected using four levels of security features:

- The eNVM page protection uses two levels: factory lock and user lock. Factory lock is not accessible for the user. Refer to the [Set Lock Bit and User Unlock Commands](#), page 153.
- There are two or four special sectors per eNVM array that can be protected for read and write, depending on which entity is accessing the region as shown in [Figure 82](#), page 160 through [Figure 86](#), page 162. On devices with smaller or bigger eNVMs, the upper 4 KB special sector is aligned to the top 4 KB region of the eNVM. These user-protectable 4 KB special sectors can be configured by Libero software, see [Figure 93](#), page 172.
- There are two private regions in M2S060, M2S090, and M2S150 as shown in [Figure 85](#), page 161 and [Figure 86](#), page 162 which are reserved for storing device certificate, eNVM digest, security keys and so on. Only system controller can access the private regions. See [eNVM Pages for Special Purpose Storage](#), page 163

- Using AHB bus master access control, the eNVM can be protected from different masters connected on the AHB bus matrix. Refer to the [AHB Bus Matrix](#), page 210.
- User-defined regions can be protected from the FPGA fabric.

5.3.1 User Protectable 4K Regions

Figure 82 • eNVM Special Sectors for the M2S050TS Device with 256 KB eNVM_0

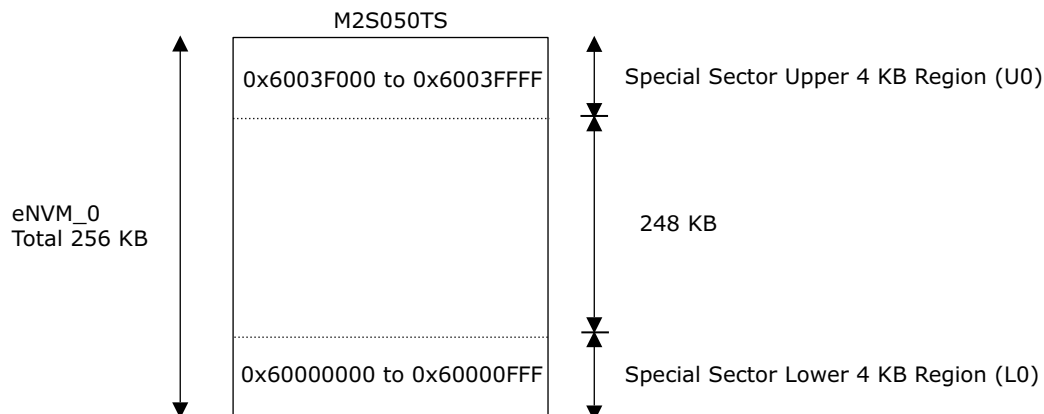


Figure 83 • eNVM Special Sectors for the M2S005S Device with 128 KB eNVM_0

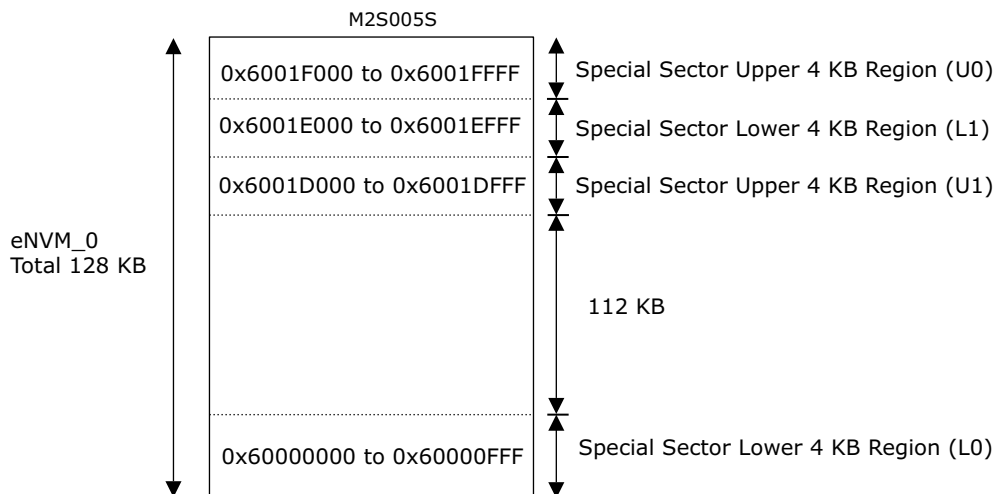


Figure 84 • eNVM Special Sectors for the M2S010TS, M2S025TS Devices with 256 KB eNVM_0

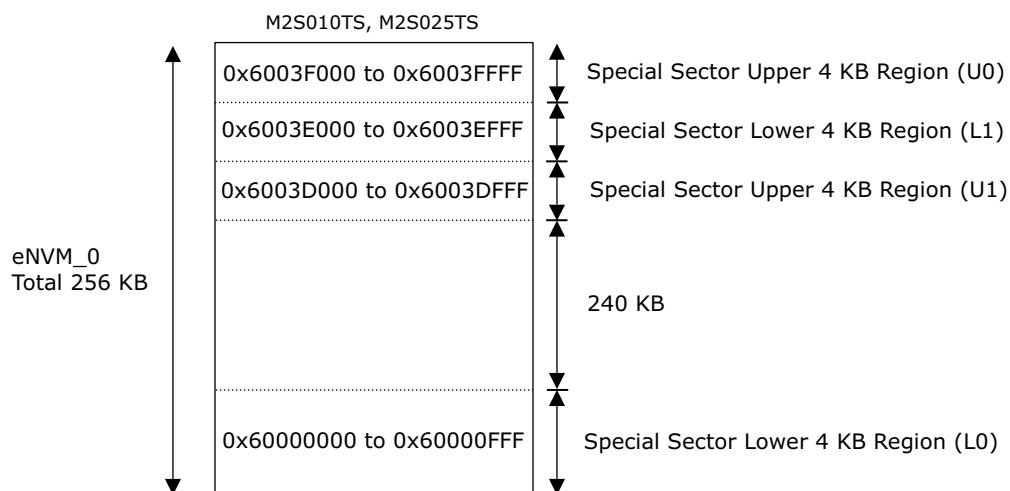


Figure 85 • eNVM Special Sectors for the M2S060TS Devices with 256 KB eNVM_0

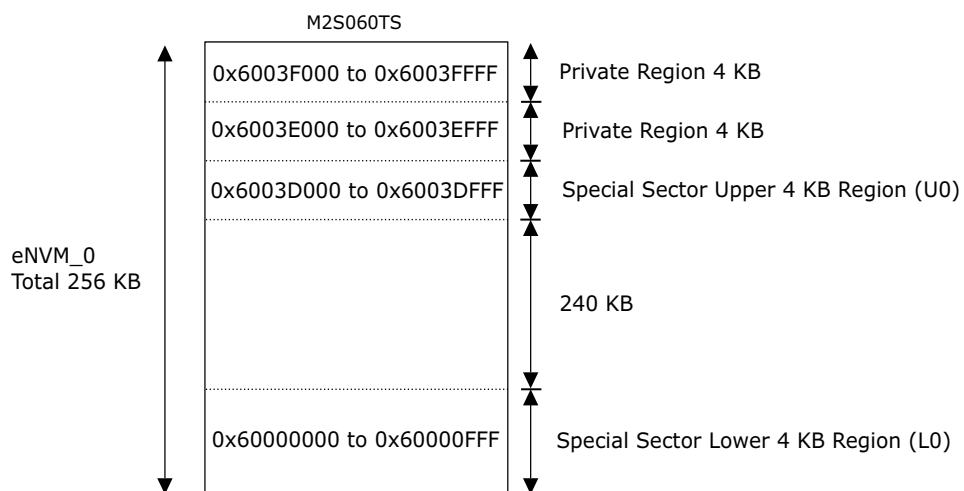
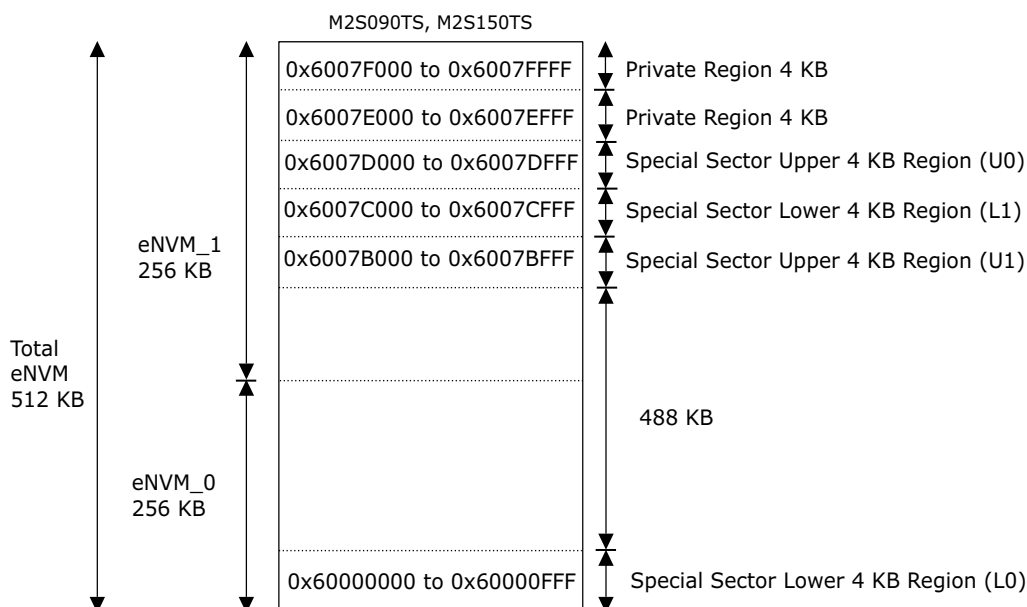


Figure 86 • eNVM Special Sectors for the M2S090TS, M2S150TS Devices with 512 KB

The security configuration is provided as input to the eNVM Controller from system registers as per the [ENVM_PROTECT_USER](#) register described in [Table 103](#), page 174 for configuration of upper and lower regions of NVM. The following table shows user protection regions for different masters.

Table 99 • User Protection Regions

Master	Function
Cortex-M3 processor	Cortex-M3 processor can access the protected memory regions. Access bit defines the read accessibility. Write allowed bit indicates that the masters which have read access can also have write access.
Fabric master	FIC_0 can access the protected memory regions. Access bit defines the read accessibility. Write allowed bit indicates that the masters which have read access can also have write access.
Other masters (PDMA and HPDMA)	All other masters are allowed access. Access bit defines the read accessibility.

5.3.1.1 Read Protection

When AHB masters other than the system controller issue read transactions to protected regions, the address and protection configuration is checked to determine whether the read is targeted to the protected region and if the read is allowed. If the read is not allowed, eNVM read command is not sent to the eNVM and an error is generated. For a specific AHB master to read a protected region, both the factory and user allowed bits must be set. Refer to [Table 108](#), page 178 for information on eNVM access controls for AHB masters.

5.3.1.2 Write Protection

When AHB masters other than system controller issue write transactions (which may be one of the program commands supported by this interface) to protected regions, the address and protection configuration is checked to determine whether the transaction is targeted to the protected region. If the transaction is not allowed, no command is sent to eNVM and the Status bit is asserted.

5.3.1.3 Power-Down

During device startup, the eNVM(s) will be powered up as the fabric is powered up. As soon as the fabric is active, if the user sets the deep power down (DPD) bit, the NVM(s) will be powered down. Each eNVM block can be put into deep power down mode by configuring the SYSREG. The eNVM can permanently be switched on or switched off. Refer to the ENVM_CR register (Table 103, page 174) for configuration settings.

During Flash*Freeze, users may want to put the NVM(s) into deep power down mode, to save power. The user should not enter power down while the NVM is in use. DPD is not entered automatically when Flash*Freeze is entered.

Note: Flash*Freeze applies mainly to the fabric.

5.3.2 eNVM Pages for Special Purpose Storage

A few pages in the final sector (N-1) of the last eNVM module are used for special purpose storage like device certificate and eNVM digest. Some special purpose pages are reserved and protected. Refer below tables for more information on eNVM special purpose storage based on SmartFusion2 device density. The system controller performs read/write operations on unreserved eNVM pages using system controller services. It only reads data from reserved eNVM pages. 16 pages in the final sector of eNVM_0 module for M2S005, M2S010, M2S025, and M2S050 devices are used for special purpose storage as listed in the following table.

Table 100 • Special Purpose Storage Regions

Device	eNVM module	Sector	Page	Type	Usage
M2S005/M2S010/ M2S025/M2S050	eNVM_0	N-1	16-24	Reserved	Reserved for future use
			25-30	Reserved	Device Certificate
			31	Unreserved	Digest for eNVM_0

64 pages of eNVM in the final 2 sectors (private regions) of the last eNVM module for M2S060, M2S090, and M2S150 devices are used as special purpose storage. See the following table for more information.

M2S060 device has 2 private regions in eNVM_0 and M2S090/M2S150 device has 2 private regions in eNVM_1.

Table 101 • Special Purpose Storage Regions for M2S060, M2S090, and M2S150 Devices

Sector in eNVM	Page	Type	Usage	Offset in page (Bytes)	Range (Bytes)
N-2	20-0	Unreserved	User Key Code#2 to User Key Code #N. N can be maximum 58. Maximum 56 Key Codes (KC#2 to KC#58), each occupies 48 Bytes Minimum 5 Key Codes (KC#2 to KC#7), each occupies 528 Bytes	0	2687:0
	29-21	Unreserved	User Activation Code	0	1151:0
	30	Unreserved	User Activation Code (Total 1192 bytes across page 21 to page 30)	0	39:0
	30	Unreserved	User Defined (Key sizes + Exported bit + Valid bit) byte array: 56 bytes holds 56 key sizes along with exported and valid bit flags.	40	55:0
	30	Unreserved	Reserved for future use	96	31:0
	31	Unreserved	User PK-X (384-bit User PUF ECC Public Key)	0	47:0
	31	Unreserved	User PK-Y (384-bit User PUF ECC Public Key)	48	47:0
	31	Unreserved	User Activation Code exported flag (Digests Valid, Activation Code missing)	96	1 byte
	31	Unreserved	User Activation Code valid flag	97	1 byte
	31	Unreserved	User Key Code #0 exported flag (Digests Valid, Key Code missing)	98	1 byte
	31	Unreserved	User Key Code #0 valid flag	99	1 byte
	31	Unreserved	User Key Code #1 exported flag (Digests Valid, Key Code missing)	100	1 byte
	31	Unreserved	User Key Code #1 valid flag	101	1 byte
	31	Unreserved	User Public Key valid flag	102	1 byte
	31	Unreserved	Reserved for future use	103	24:0

Table 101 • Special Purpose Storage Regions for M2S060, M2S090, and M2S150 Devices (continued)

N-1	0	Unreserved	User Key Code #0 (256-bit User AES Key)	0	43:0
	0	Unreserved	User Key Code#1 (384-bit User PUF ECC Key) (76 bytes)	44	75:0
	0	Unreserved	Reserved for future use	120	7:0
	9-1	Reserved	Factory Activation Code	0	1151:0
	10	Reserved	Factory Activation Code (Total 1192 bytes across page 1 to page 10)	0	1191:1152
	10	Reserved	Factory Key Code (384 bit Factory ECC Key Code)	40	75:0
	10	Reserved	Reserved for future use	116	11:0
	15-11	Reserved	Second ECC Key Certificate	0	639:0
	21-16	Reserved	Reserved for future use	0	767:0
	22	Unreserved	eNVM_1 Private User Digest of page 0 of N-1 and all pages of N-2	0	127:0
	23	Reserved	eNVM_1 Private Factory Digest of pages from 1 to 30 of N-1 except pages 22, 23, and 24	0	127:0
	24	Unreserved	eNVM_1 Public Digest	0	127:0
	30-25	Reserved	Device Certificate	0	767:0
	31	Unreserved	eNVM_0 Digest	0	127:0

Notes:

- Refer to [UG0443: SmartFusion2 SoC FPGA and IGLOO2 FPGA Security and Reliability User Guide](#) for more information on the certificates, key codes, and digests.
- The system controller performs read/write operations on unreserved eNVM pages using system controller services. It only reads data from reserved eNVM pages.

5.4 How to Use eNVM

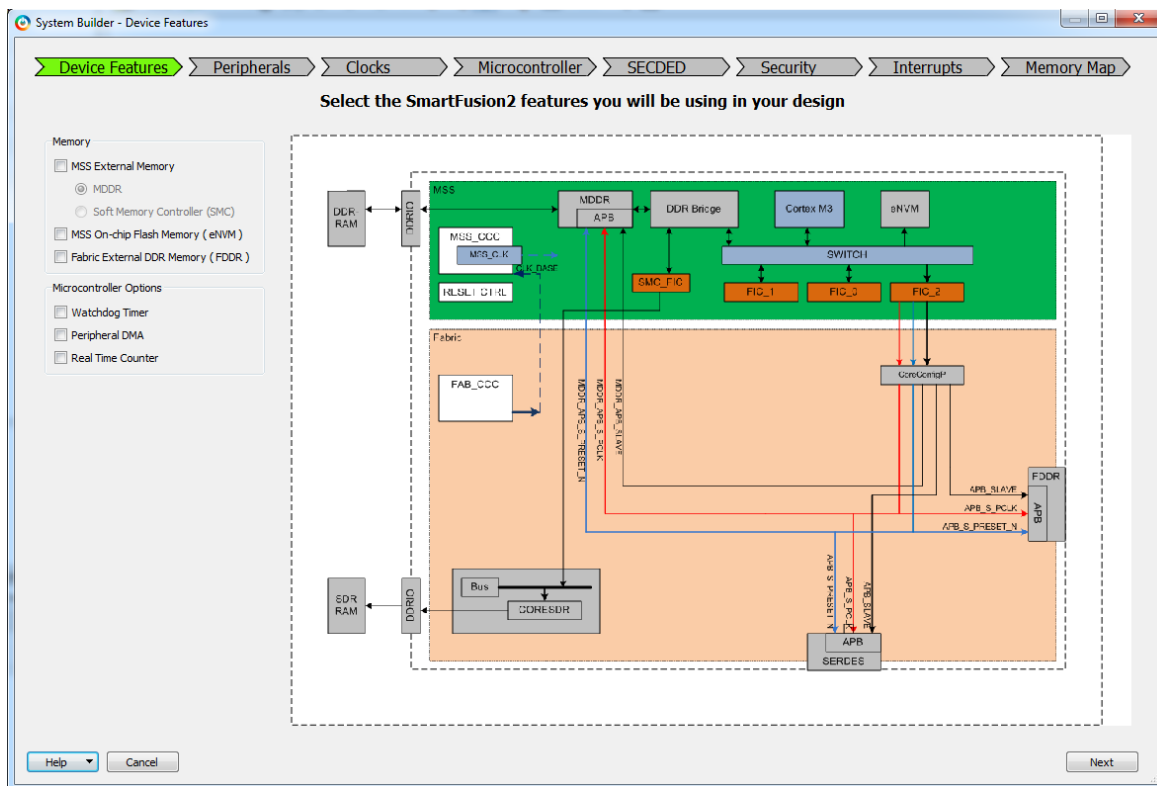
This section describes how to use the eNVM in the SmartFusion2 devices. To configure the SmartFusion2 device features and then build a complete system, use the **System Builder** graphical design wizard in the Libero SoC software.

5.4.1 Data Storage in eNVM Using the Libero eNVM Client

The Libero eNVM client creates the eNVM data that the FlashPro software uses to initialize the eNVM during programming. The programmed eNVM can be accessed by the Cortex-M3 processor, HPDMA, PDMA, or the FPGA fabric master connected to the AHB bus matrix.

The following figure shows the initial **System Builder** window where the required device features can be selected. For details on how to launch the **System Builder** wizard and detailed information on how to use it, refer to [SmartFusion2 System Builder User Guide](#).

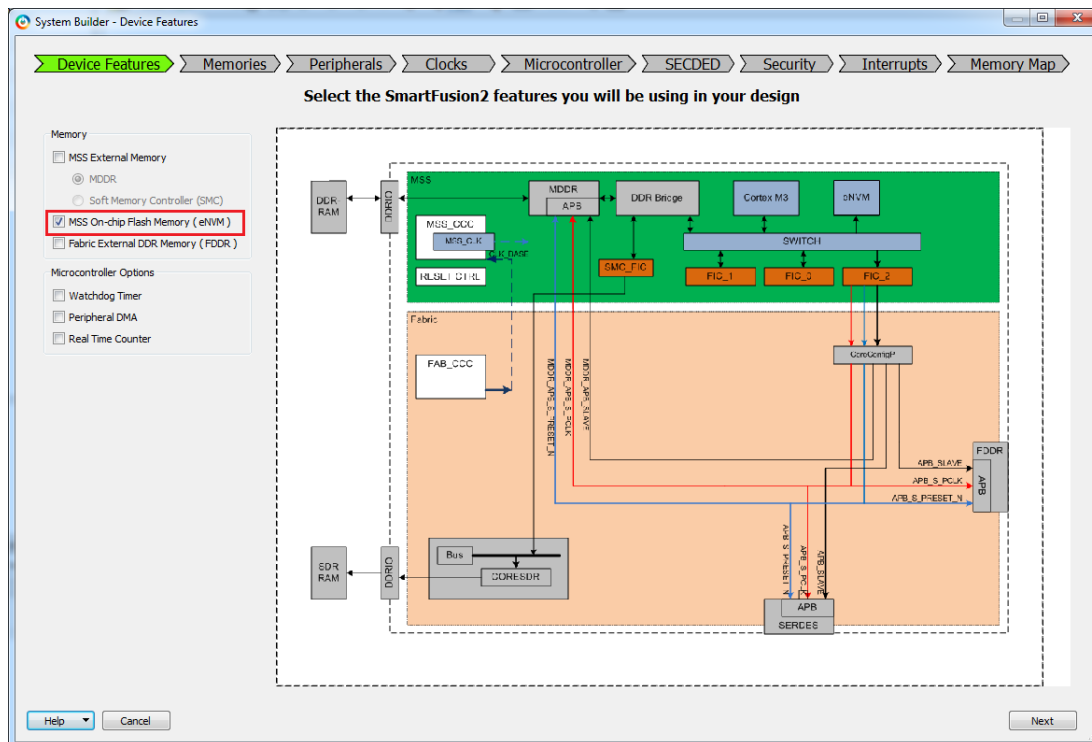
Figure 87 • System Builder Window



The following steps describe how to generate a programming file with the eNVM client in an application using System Builder.

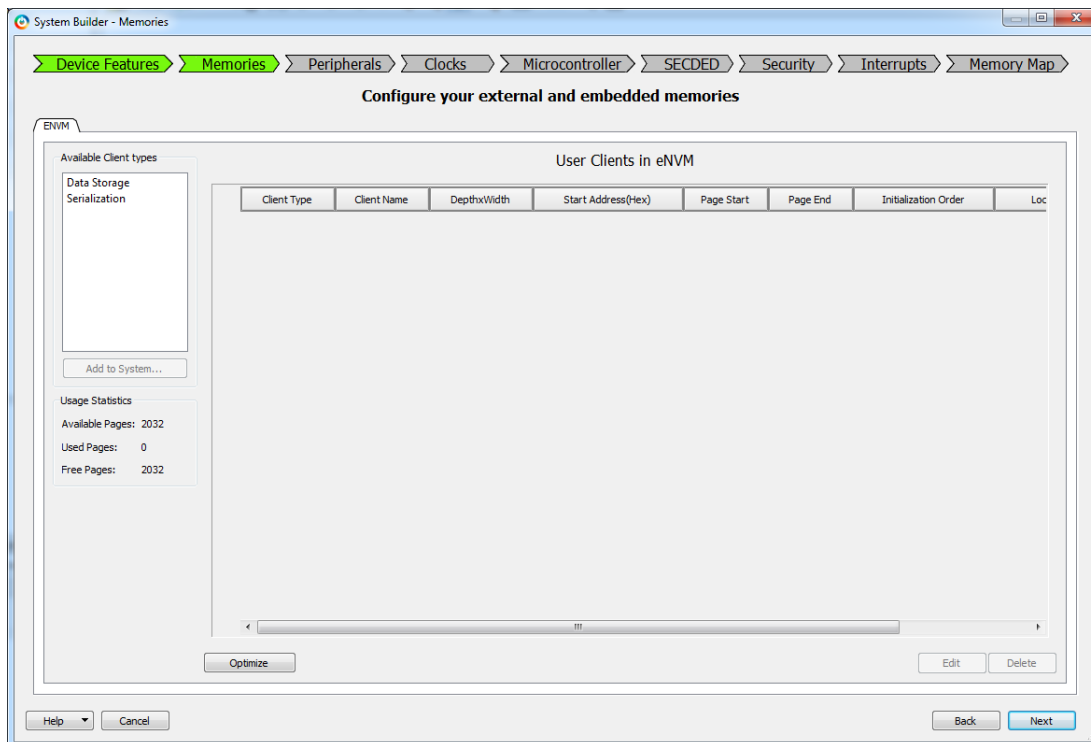
1. Check the **MSS On-chip Flash Memory (eNVM)** check box under the **Device Features** tab and leave the other check boxes unchecked. The following figure shows the **System Builder - Device Features** tab.

Figure 88 • System Builder - Device Features Tab



- Click **Next** to navigate to the **Memories** tab. The following figure shows the **System Builder - Memories** tab.

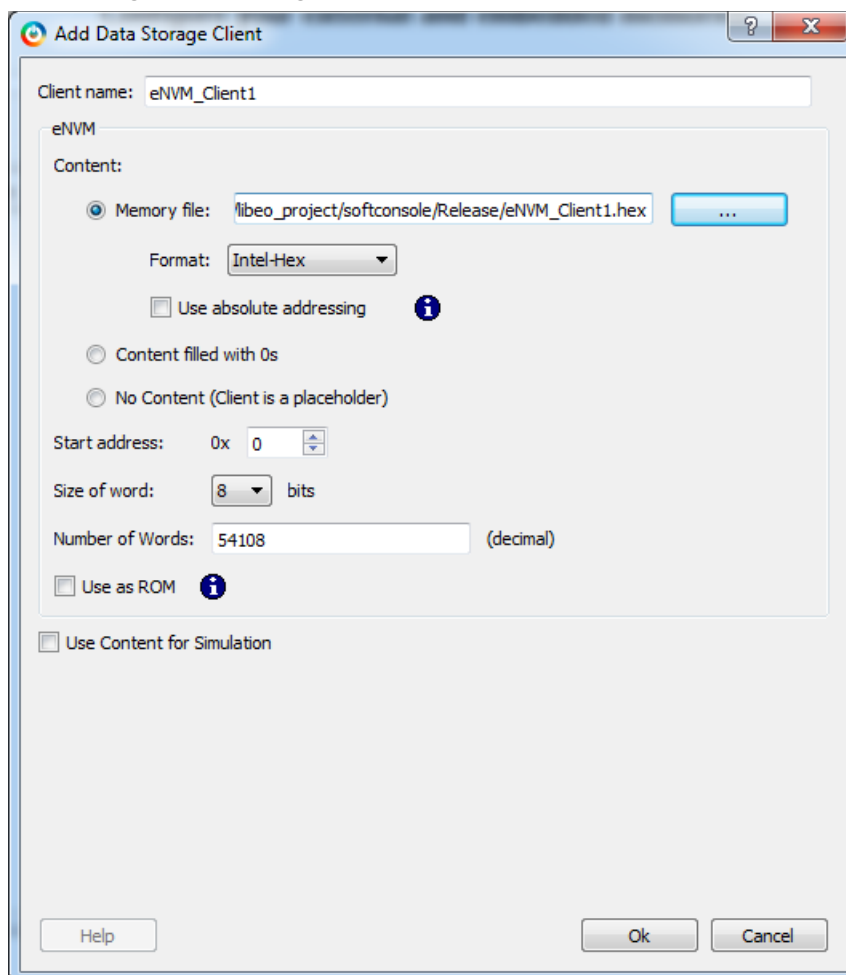
Figure 89 • System Builder - Memories Tab



- Select **Data Storage** under **Available Client Types** and click **Add to System**. The following figure shows the **Add Data Storage Client** dialog. It supports the following file formats:
 - Intel-Hex
 - Motorola-S
 - Microsemi-Hex
 - Microsemi-Binary
- Create the memory file in any one of the above formats with the executable code or data. Memory file can be created for the code using the SoftConsole 3.4 or later with the linker script "production-execute-in-place.ld". For more details on using the SoftConsole, refer to the [SoftConsole User Guide](#).

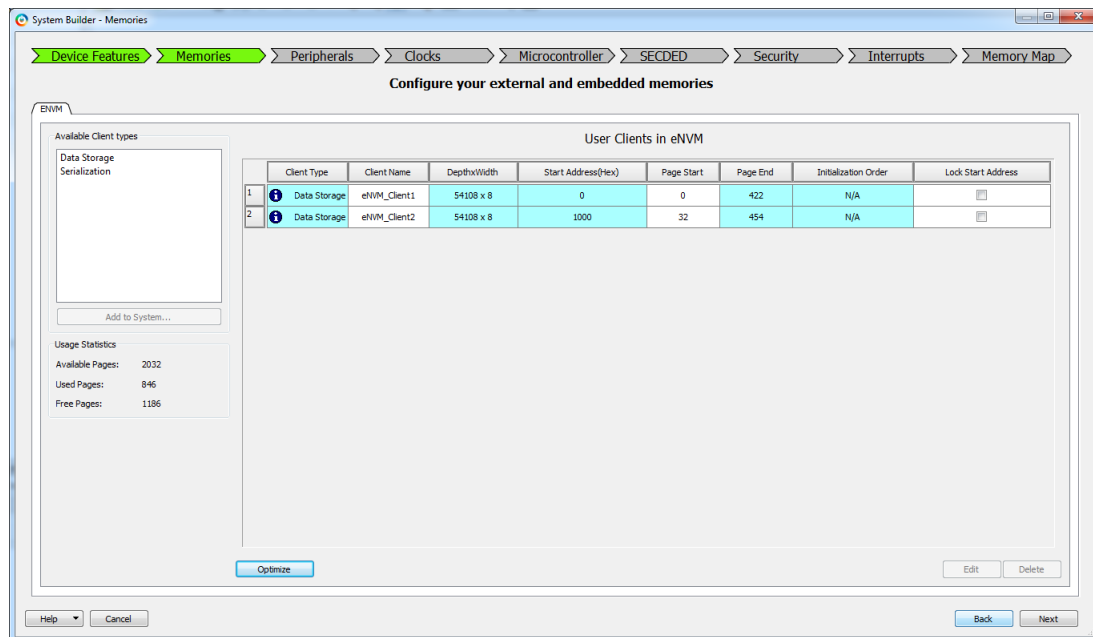
5. Enter the Client name, navigate to the memory file location using **Browse**, and select it. Give the rest of the parameters according to the requirements and click **OK** to add the eNVM client. For more information on **Use absolute addressing**, **Use as ROM** and other options, click **Help**.

Figure 90 • Add Data Storage Client Dialog



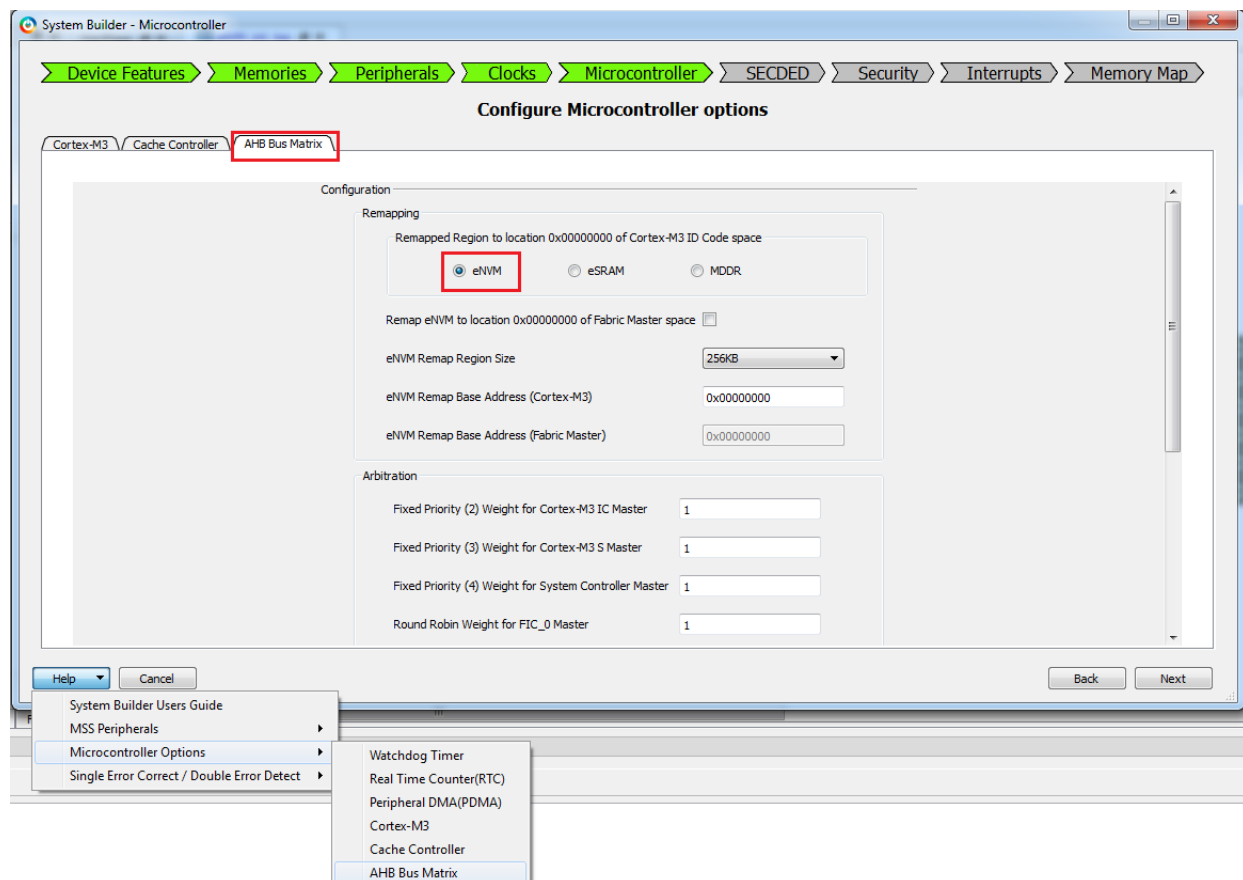
6. The eNVM client data is populated in the **System Builder - Memories** tab. The following figure shows the **System Builder - Memories** tab with two eNVM clients.

Figure 91 • System Builder - Memories Tab with Two eNVM Clients



7. Navigate to the **Microcontroller** tab in the System Builder and select **AHB Bus Matrix** to confirm the remapping of eNVM to the Cortex-M3 code space. eNVM is remapped to the Cortex-M3 code space by default. Click **Help** and select **AHB Bus Matrix** to access the help document for more information on eNVM Remap Region Size and Base Address, as shown in the following figure. Refer to *AC390: SmartFusion2 SoC FPGA – Remapping eNVM, eSRAM, and DDR/SDR SDRAM Memories Application Note*.

Figure 92 • System Builder - Microcontroller Tab

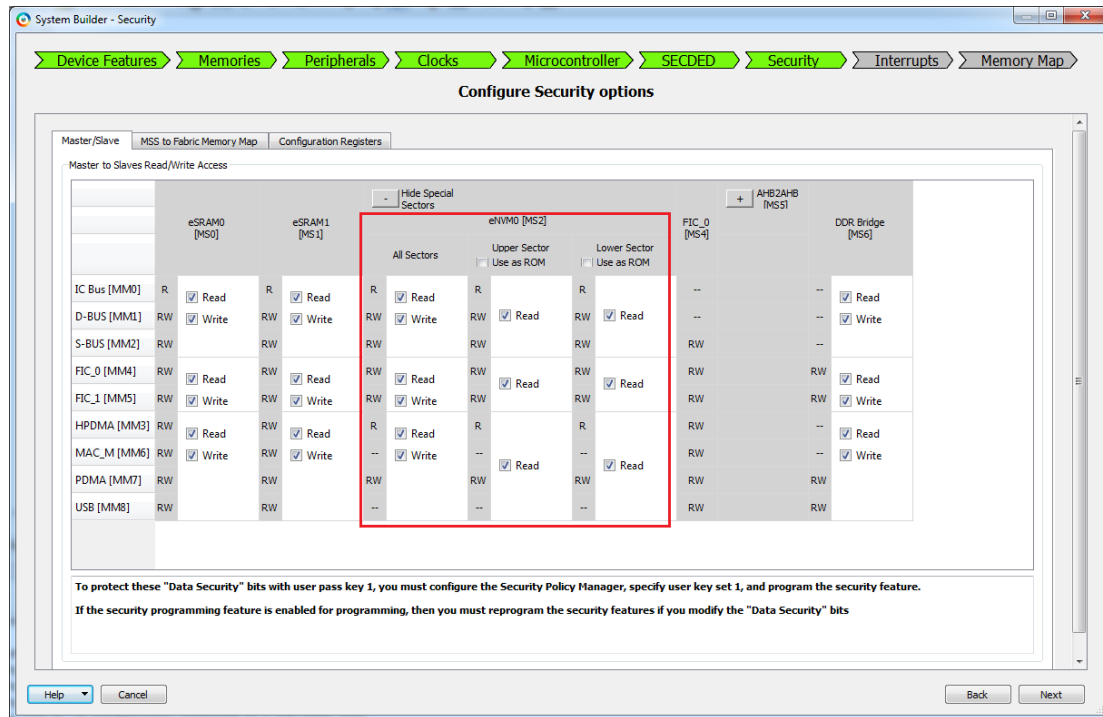


Notes:

- The code executing from eNVM can program the other regions of eNVM memory. Ensure that the code executing region must not be overwritten.
- If the user design is using the FPGA fabric based master, the Cortex-M3 processor requires a valid program in eNVM (from eNVM start address 0x60000000) to execute at power-up or power-on reset. The valid program can be a simple user boot code or a simple loop program. You can select a .hex file of a valid program for eNVM data client using the SystemBuilder.

- Navigate to the **Security** tab to select the read and write access permissions of eNVM including protected regions for different masters as shown in the following figure.

Figure 93 • System Builder - Security Tab



The read and write permission options for different masters are available for data and design security enabled devices like M2S050TS only.

For more information on configuring the security options, refer to [SmartFusion2 MSS Security Configuration](#).

- Navigate to the **Memory Map** tab giving the required data in the rest of the **System Builder** tabs and click **Finish** to proceed with creating the MSS Subsystem.
- Do required Pin connections and Save the project. Generate the SmartDesign in Libero by clicking **Generate Component**.
- Double-click **Run PROGRAM Action** in the Libero **Design Flow** window to program the SmartFusion2 device to initialize the eNVM with the memory file.

Notes:

- The MSS eNVM supports full behavioral simulation models. Refer to [SmartFusion2 MSS Embedded Nonvolatile Memory \(eNVM\) Simulation User Guide](#) for information on how to simulate the eNVM operations.
- Refer to the [AC429: SmartFusion2 and IGLOO2 - Accessing eNVM and eSRAM from FPGA Fabric Application Note](#) for information on how to access the eNVM using FPGA fabric logic.
- Refer to the [AC426: Implementing Production Release Mode Programming for SmartFusion2 Application Note](#) for more information on how to add multiple eNVM data storage clients using the Libero SoC software.

5.4.2 Reading the eNVM Block

Any master connected on the AHB bus matrix (for example, Cortex-M3 processor, HPDMA, PDMA, user logic in FPGA) can access the eNVM blocks using the address range provided in [Table 93 on page 146](#) for read operations.

5.4.3 Writing to the eNVM Block

Writing to eNVM by using the Cortex-M3 processor can be done by the API provided in the eNVM driver in the Firmware Catalog. FPGA fabric user logic can implement the state machine to write into the eNVM by implementing the commands sequence explained in the eNVM commands section.

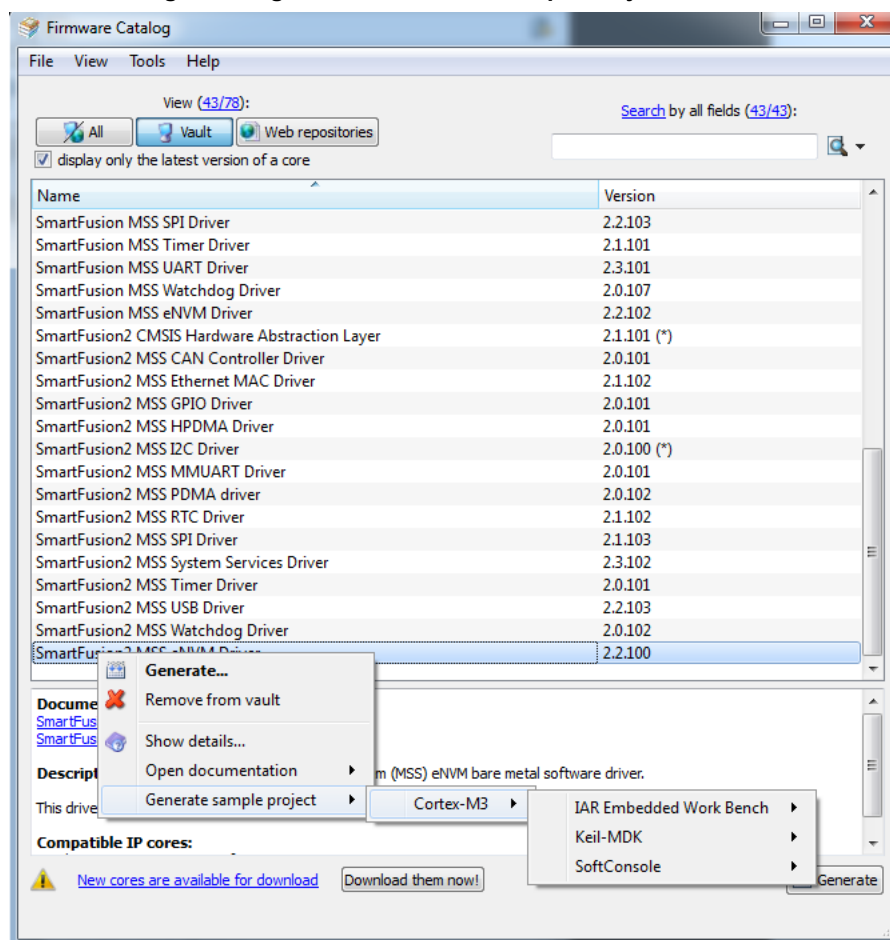
Refer to [AC391: SmartFusion2 SoC FPGA - eNVM Initialization Application Note](#) for information about reading and writing to the eNVM block using the Cortex-M3.

5.4.4 Firmware and Sample Project

Microsemi provides eNVM firmware drivers to use with the application development. The SmartFusion2 eNVM firmware drivers can be downloaded from the Firmware Catalog. The eNVM firmware driver provides APIs to unlock and write to eNVM features. Refer to the **SmartFusion2 eNVM Driver User Guide** from **Open Documentation** for the list of APIs and their descriptions.

The eNVM driver package includes sample projects to show the usage of eNVM. The sample projects are available for three different tool chains: IAR Embedded Work, Keil-MDK, and SoftConsole. The sample project can be generated by right-clicking the eNVM driver and selecting **Generate sample project**, as shown in the following figure.

Figure 94 • Firmware Catalog Showing the Generation of Sample Project for eNVM



The following table lists the available APIs for eNVM in the eNVM firmware drivers

Table 102 • Available APIs for eNVM

API	Description
NVM_unlock	Unlock the eNVM Block
NVM_write	The function NVM_write() is used to program data in to the eNVM. This function treats the two eNVM blocks contiguously, hence 512 KB of memory can be accessed linearly. The starting address and ending address of the memory to be written need not be page aligned. This function supports programming data that spans across multiple pages, and does not support writing or programming eNVM without input data. This function is a blocking function. The NVM_write() function performs a verify operation on each page programmed to ensure the NVM is programmed with the expected content.

5.5 SYSREG Control Registers

The System Control registers control eNVM behavior. These registers are located in the SYSREG section and are listed here for clarity. Refer to the [System Register Block](#), page 670 for more information on each register and bit.

Table 103 • SYSREG Control Registers

Register Name	Register Type	Flash Write Protect	Reset Source	Description
ENVM_CR (0x4003800C)	RW-P	Register	sysreset_n	eNVM Configuration register.
ENVM_REMAP_BASE_CR (0x40038010)	RW-P	Register	sysreset_n	eNVM remap Configuration register for the Cortex-M3 processor.
ENVM_REMAP_FAB_CR (0x40038014)	RW-P	Register	sysreset_n	eNVM remap configuration register for a soft processor in the FPGA.
ENVM_PROTECT_USER (0x40038144)	RO-U	N/A	sysreset_n	Configuration for accessibility of protected regions of eNVM_0 and eNVM_1 by different masters on the AHB bus matrix. This register gets updated by flash bit configuration set during device programming. This configuration can be done through the System Builder using settings on the Security tab.
ENVM_STATUS (0x40038148)	RO-U	N/A	sysreset_n	Code shadow Status register.
ENVM_SR (0x40038158)	RO	N/A	sysreset_n	Indicates busy status for eNVM_0, eNVM_1.

Table 104 • ENVN_CR

Bit Number	Name	Reset Value	Description
[31:17]	Reserved	0	
16	ENVN_SENSE_ON	0	Turns on or off the sense amps for both NVM0 and NVM1. The sense amp switching feature is useful to decrease the eNVM access time. 0: Normal Operation -The sense amp switches off after every read cycle if an idle cycle follows. This saves power but slightly increases access time on the next read cycle. 1:The sense amp is turned ON. This increases power but decreases access times.
15	ENVN_PERSIST	0	Reset control for NVM0 and NVM1. 0: NVM0, NVM1 will get reset on SYSRESET_N and PORESET_N. 1: NVM0, NVM1 will get reset on PORESET_N.
14	NV_DPD1	0	Deep power-down control for the NVM1. 0: Normal operation 1: NVM deep power-down
13	NV_DPD0	0	Deep power-down control for the NVM0. 0: Normal operation 1: NVM deep power-down

Table 104 • ENVM_CR (continued)

Bit Number	Name	Reset Value	Description
[12:5]	NV_FREQRNG	0x7	<p>Setting of NV_FREQRNG[8:5] or NV_FREQRNG[12:9] determines the behavior of eNVM BUSY_B with respect to the AHB Bus interface clock. It can be used to accommodate various frequencies of the external interface clock, M3_CLK, or it can be used to advance or delay the data capture due to variation of read access time of the NVM core. It sets the number of wait states to match with the Cortex-M3 or Fabric master operating frequency for read operations. The small counter in the NVM Controller uses this value to advance or delay the data capture before sampling data.</p> <p>0000: NOT SUPPORTED 0001: NOT SUPPORTED 0010: Page Read = 3, All other modes (Page program and Page verify) = 2 0011: Page Read = 4, All other modes (Page program and Page verify) = 2 0100: Page Read = 5, All other modes (Page program and Page verify) = 2 0101: Page Read = 6, All other modes (Page program and Page verify) = 3 0110: Page Read = 7, All other modes (Page program and Page verify) = 3 0111: Page Read = 8, All other modes (Page program and Page verify) = 4 1000: Page Read = 9, All other modes (Page program and Page verify) = 4 1001: Page Read = 10, All other modes (Page program and Page verify) = 4 1010: Page Read = 11, All other modes (Page program and Page verify) = 5 1011: Page Read = 12, All other modes (Page program and Page verify) = 5 1100: Page Read = 13, All other modes (Page program and Page verify) = 6 1101: Page Read = 14, All other modes (Page program and Page verify) = 6 1110: Page Read = 15, All other modes (Page program and Page verify) = 6 1111: Page Read = 16, All other modes (Page program and Page verify) = 7</p> <p>NV_FREQRNG[8:5] is used for NVM0 and NV_FREQRNG[12:9] is used for NVM1.</p>
4:0	SW_ENVMREMAPSIZE	0x11	<p>Size of the segment in eNVM, which is to be remapped to location 0x00000000. This logically splits eNVM into a number of segments, each of which may be used to store a different firmware image, for example. The region sizes are shown in Table 105, page 177.</p>

Table 105 • SW_ENVMREMAPSIZE

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Remap Size
0	0	0	0	0	Reserved
0	0	0	0	1	Reserved
0	0	0	1	0	Reserved
0	0	0	1	1	Reserved
0	0	1	0	0	Reserved
0	0	1	0	1	Reserved
0	0	1	1	0	Reserved
0	0	1	1	1	Reserved
0	1	0	0	0	Reserved
0	1	0	0	1	Reserved
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	Reserved
0	1	1	0	1	16 KB
0	1	1	1	0	32 KB
0	1	1	1	1	64 KB
1	0	0	0	0	128 KB
1	0	0	0	1	256 KB
1	0	0	1	0	512 KB, reset value

Table 106 • ENVM_REMAP_BASE_CR

Bit Number	Name	Reset Value	Description
[31:19]	Reserved	0	Reserved.
[18:1]	SW_ENVMREMAPBASE	0	Offset address of eNVM for remapping. SW_ENVMREMAPBASE indicates the offset within eNVM address space of the base address of the segment in eNVM, which is to be remapped to location 0x00000000. Bit 0 of this register is defined as SW_ENVMREMAPENABLE and must be set to get the remapping done with new addresses filled in this register.
0	SW_ENVMREMAPENABLE	0	0: eNVM remap not enabled. Bottom of eNVM is mapped to address 0x00000000. 1: eNVM remap enabled. eNVM visible at 0x00000000 is a remapped segment of the eNVM.

Table 107 • ENVM_REMAP_FAB_CR

Bit Number	Name	Reset Value	Description
[31:19]	Reserved	0	
[18:1]	SW_ENVMFABREMAPBASE	0	Offset within eNVM address space of the base address of the segment in eNVM, which is to be remapped to location 0x00000000 for use by a soft processor in the FPGA fabric. The base address of the remapped segment of eNVM is determined by the value of this register. Bit 0 of this register is defined as SW_ENVMFABREMAPENABLE. Bit 0 must be set to remap the NVM.
0	SW_ENVMFABREMAPENABLE	0	0: eNVM fabric remap not enabled for access by fabric master/soft processor. The portion of eNVM visible in the eNVM window at location 0x00000000 of a soft processor's memory space corresponds to the memory locations at the bottom of eNVM. 1: eNVM fabric remap enabled. The portion of eNVM visible at location 0x00000000 of a soft processor's memory space of is a remapped segment of eNVM.

Table 108 • ENVM_PROTECT_USER

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	
15	NVM1_UPPER_WRITE_ALLOWED	0x1	When set indicates that the masters who have read access can have write access to the upper protection region of eNVM1. This is updated by the user flash row bit.
14	NVM1_UPPER_OTHERS_ACCESS	0x1	When set indicates that the other masters can access the upper protection region of eNVM1. This is set by the user flash row bit.
13	NVM1_UPPER_FABRIC_ACCESS	0x1	When set indicates that the fabric can access the upper protection region of eNVM1. This is set by the user flash row bit.
12	NVM1_UPPER_M3ACCESS	0x1	When this bit is set, it indicates that the Cortex-M3 processor can access the upper protection region of eNVM1. This is updated by the user flash row bit.
11	NVM1_LOWER_WRITE_ALLOWED	0x1	When set indicates that the masters who have read access can have write access to the lower protection region of eNVM1. This is set by the user flash row bit.
10	NVM1_LOWER_OTHERS_ACCESS	0x1	When set indicates that the other masters can access the lower protection region of eNVM1. This is set by the user flash row bit.
9	NVM1_LOWER_FABRIC_ACCESS	0x1	When set indicates that the fabric can access the lower protection region of eNVM1. This will be set by user flash row bit.

Table 108 • ENVM_PROTECT_USER (continued)

Bit Number	Name	Reset Value	Description
8	NVM1_LOWER_M3ACCESS	0x1	When this bit is set, it indicates that the M3 can access the lower protection region of eNVM1. This will be set by the user flash row bit.
7	NVM0_UPPER_WRITE_ALLOWED	0x1	When set indicates that the masters who have read access can have write access to the upper protection region of eNVM0. This will be set by the user flash row bit.
6	NVM0_UPPER_OTHERS_ACCESS	0x1	When set indicates that the other masters can access the upper protection region of eNVM0.
5	NVM0_UPPER_FABRIC_ACCESS	0x1	When set indicates that the fabric can access the upper protection region of eNVM0. This will be set by the user flash row bit.
4	NVM0_UPPER_M3ACCESS	0x1	When this bit is set, it indicates that the M3 can access the upper protection region of eNVM0. This will be set by the user flash row bit.
3	NVM0_LOWER_WRITE_ALLOWED	0x1	When set indicates that the masters who have read access can have write access to the lower protection region of eNVM0. This will be set by the user flash row bit.
2	NVM0_LOWER_OTHERS_ACCESS	0x1	When set indicates that the other masters can access the lower protection region of eNVM0. This will be set by the user flash row bit.
1	NVM0_LOWER_FABRIC_ACCESS	0x1	When set indicates that the fabric can access the lower protection region of eNVM0. This will be set by the user flash row bit.
0	NVM0_LOWER_M3ACCESS	0x1	When this bit is set, it indicates that the M3 can access the lower protection region of eNVM0. This will be set by the user flash row bit.

Note: Refer to Table [Table 99](#), page 162 for information on different masters.

Table 109 • ENVM_STATUS

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	CODE_SHADOW_EN	0	Read by the system controller during device start-up, to indicate whether the user has configured the device such that code shadowing is to be performed by system controller firmware.

Table 110 • ENV_MSR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	
[1:0]	ENV_MSR_BUSY	0	Active high signals indicate a busy state per eNVM for CLK-driven operations and for internal operations triggered by the write/program/erase/transfer command. ENV_MSR_BUSY[1] = Busy indication from ENV_M1 ENV_MSR_BUSY[0] = Busy indication from ENV_M0

5.6 eNVM Control Registers

To perform any transaction with the NVM array, the Control registers must be configured appropriately as per [Table 99](#), page 162.

To access or update the Control register, the AHBL master must first get access to the register set. Without access rights, all writes to the Control register will be ignored and the read will return zero from REQACC and the [Status Register Bit Definitions](#).

This access rights system ensures that while a master is programming the NVM array, no other master can interfere or see what data is being programmed.

To obtain access rights, the master writes 0x1 to the REQACC register and then reads the register to check whether access is granted. If access is granted the Control register is set.

The following table shows the base address of the eNVM Control registers for eNVM_0 and eNVM_1.

Table 111 • eNVM Control Registers Base Address

eNVM Block	Control Registers Base Address
eNVM_0	0x60080000
eNVM_1	0x600C0000

Table 112 • Control Registers Description

OFFSET HADDR[8:0]	Register Name	Width	Type	Default	Access Rights	Description
0x000-0x07F	Assembly Buffer	1023:0 32 × 32bits	R		Exclusive access to the requested master	Reads from these address will return data read from assembly buffer within the NVM array.
0x080-0x0FF	WDBUFF (Write Data Buffer)	1023:0 32 × 32bits	R/W	0	Any master on AHB bus matrix	Write data buffer This register is cleared when exiting normal mode. This register is not cleared when the System Controller grabs ownership by writing 0x03 to REQACCESS.
0x120	Status	31:0	R		Any master on AHB bus matrix	Refer to Table 113 , page 184.

Table 112 • Control Registers Description (continued)

OFFSET HADDR[8:0]	Register Name	Width	Type	Default	Access Rights	Description
0x128	NV_PAGE_STATUS	1:0	R/W	0	Exclusive access to the requested master	Refer to Table 115 , page 185.
0x12C	NV_FREQRNG[7:0]	7:0	R	SYSREG	Exclusive access to the requested master	<p>eNVM interface frequency range setting: Bits [3:0] set the number of wait cycles required for each NVM access cycles. This is read-only register. The ENVM_CR system register NV_FREQRNG field needs to be set with value as calculated below. $NV_FREQRNG = \text{roundup}(40 \text{ ns} / M3_CLK \text{ clock period in ns})$ The NV_FREQRNG[3:0] is for NVM0 wait states and NV_FREQRNG[7:4] is for NVM1 wait states. Refer to Table 114 on page 185 NV_FREQRNG calculations at different M3_CLK frequencies for all SmartFusion2 devices. Bits [7:4] are unused with the AHB-NVM block when the device has only eNVM_0. This controls the NV_FREQRNG[3:0] input on the NVMCTRL function that sets the required number of clock cycles required for NVM accesses relative to the operating frequency.</p>
0x130	NV_DPD_B	1-bit	R	SYSREG	Exclusive access to the requested master	<p>NV_DPD_B[0] describes NVM deep power-down state. 0: NVM operational 1: NVM In deep power-down</p>

Table 112 • Control Registers Description (continued)

OFFSET HADDR[8:0]	Register Name	Width	Type	Default	Access Rights	Description
0x134	NV_CE	2-bit	R/W	1	Exclusive access to the requested master	NV_CE[0] = 0: NVM disabled NV_CE[0] = 1: NVM enabled NV_CE [1] = 1; The internal read cache is disabled. All reads will directly read the eNVM array, or AB space. When set NVM access latency will increase. By default this bit is set to '0'.
0x140	PAGE_LOCK_SET	1	R/W	0	Exclusive access to the requested master	PAGE_LOCK_SET[0] = 1: Page is locked. PAGE_LOCK_SET[0] = 0: Page is unlocked. If the page is locked, then before writing the page should be unlocked.
0x144	DWSIZE	3:0	R/W	0	Exclusive access to the requested master	Write size in number of double words, to be written to assembly buffer from Write Data buffer during NVM commands. See description for individual commands. 0000 = 1 dword 1111 = 16 dwords
0x148	CMD	31:0	R/W	0	Exclusive access to the requested master	Write to CMD and if command field in HWDATA decoded to be a command, then NVM command will be initiated. See description of Table 103 , page 174: CMD register and individual commands.
0x154	INTEN[10:0]	10:0	R/W	0	Exclusive access to the requested master	Writing '1' to each bit will enable the corresponding interrupt.
0x158	CLRHint[2:0]	2:0	W	0	Exclusive access to the requested master	Clear interrupts/flag/busy bit by writing 1 to the corresponding bit.

Table 112 • Control Registers Description (continued)

OFFSET HADDR[8:0]	Register Name	Width	Type	Default	Access Rights	Description
0x1FC	REQACCESS	2:0	R/W	000	Any master on AHB bus matrix. This register can only be accessed using word, half or byte accesses to address 0x01FC. Accesses to addresses 0x1FD, 0x1FE, and 0x1F should not be used.	<p>Request register access. When written with 0x01, it will request exclusive access.</p> <p>Read indicates whether access has granted or not or which entity currently has been granted access.</p> <p>Read Value [2:0]</p> <p>0XX: No entity has access</p> <p>The XX value indicates who had last access.</p> <p>100: System controller</p> <p>101: M3</p> <p>110: Fabric</p> <p>111: Other master (such as PDMA or HDMA)</p> <p>To release access rights, write 0x00.</p> <p>The System Controller may gain immediate access by writing 0x03 to this register. When access is relinquished, the WDBUFF buffer, and RDBUFF buffers are cleared.</p>

Note: Addresses that are not mentioned in the register range are either reserved or exclusively for System Controller usage.

5.6.1 Status Register Bit Definitions

Table 113 • Status Register Bit Definitions

Bit	Description
[31:29]	Value of locked state of the AHB interface. These bits contain the same information as REQACCESS[2:0].
[28:20]	Reserved
19	Command when Busy. Indicates that a command was loaded while the controller was busy and has been ignored. Once set all command operations are disabled. Cleared by writing 1 to bit-2 in CLRHint[2:0] .
18	Access denied. Indicates that read or writes operations were denied due to protection systems, or that an illegal command was loaded. Once set all command operations are disabled. Cleared by writing 1 to bit 1 in the CLRHint[2:0] .
17	NVM deep power-down state, indicates NVM has entered DPD mode 0: NVM operational 1: NVM In deep power down There is delay of ~5us for the NVM to enter power down and assert this bit from requesting power down.
[16:15]	RDBUFF3 (Read data buffer 3 = Read data buffer[255:192]) ECC status (2-bit error, 1 bit corrected) 00: no error 01: 1 bit corrected 10: 2 bit detected 11: 3 or more bits detected
[14:13]	RDBUFF2 (Read data buffer 2 = Read data buffer[191:128]) ECC status (2-bit error, 1 bit corrected) 00: no error 01: 1 bit corrected 10: 2 bit detected 11: 3 or more bits detected
[12:11]	RDBUFF1 (Read data buffer 1 = Read data buffer[127:64]) ECC status (2-bit error, 1 bit corrected) 00: no error 01: 1 bit corrected 10: 2 bit detected 11: 3 or more bits detected
[10:9]	RDBUFF0 (Read data buffer 0 = Read data buffer[63:0]) ECC status (2-bit error, 1 bit corrected) 00: no error 01: 1 bit corrected 10: 2 bit detected 11: 3 or more bits detected
8	Asserted for ECC2 (2 bit error). Valid after read and read assembly buffer.
7	Asserted for ECC1 (1 bit correction). Valid after read and read assembly buffer.
6	Asserted for refresh required. Valid after program, write only and page erase.
5	Asserted when write count is over threshold. Valid after program, verify and read page status. The threshold value per eNVM page is 1000 or 10000 depending on the data retention period. See DS0128: IGLOO2 FPGA and SmartFusion2 SoC FPGA Datasheet for more information on programming cycles and retention time.
4	Asserted for program or erase failure due to page lock. Valid after program, page erase. Asserted for write failure when executing write only on the page with currently erased page bar (CEPB) = 1.
3	Asserted for write verify failure. Valid after program and write only.
2	Asserted for erase verify failure. Valid after program, page erase.
1	Asserted for verify failure. Valid only after verify operation.

Table 113 • Status Register Bit Definitions (continued)

Bit	Description
0	NVM Ready/busy 0: Busy 1: Ready

Table 114 • NV_FREQRNG Calculations at Different M3_CLK Frequencies for All SmartFusion2 Devices

Standard MSS Frequencies							
M3_CLK in MHz	166	142	133	100	66	50	33
NV_FREQRNG[3:0]	0x7	0x6	0x 5	0x 4	0x 3	0x 2	0x 2
NV_FREQRNG[7:4]	0x7	0x6	0x 5	0x 4	0x 3	0x 2	0x 2
NV_FREQRNG[7:0]	0x77	0x66	0x55	0x44	0x33	0x22	0x22

Note: If M3_CLK is 166MHz, clock period is 6.024ns. NV_FREQRNG[3:0] = roundup(40ns/6.024ns)=7.

Table 115 • NV_PAGE_STATUS

Bit	Description
1	R/W page status select
0	Reserved

Table 116 • INTEN[10:0]

Bit	Description
10	Command loaded when busy
9	NVM command denied by protection
8	NVM internal operation (erase/program/write only) is complete
7	ECC2 (2-bit error)
6	ECC1 (1-bit correction)
5	Refresh required
4	Write count is over threshold
3	Program or erase failure due to page lock. Write failure when executing write only on the page with currently erased page bar (CEPB) =1.
2	Write verify failure
1	Erase verify failure
0	Verify failure

Table 117 • CLRHINT[2:0]

Bit	Description
2	Clear the internal command when busy bit
1	Clear the internal access denied flag
0	Clear HINTERRUPT output

6 Embedded SRAM (eSRAM) Controllers

SmartFusion2 SoC FPGAs have two embedded SRAM (eSRAM) blocks of 32 KB each for data read and write operations. These eSRAM blocks are interfaced through eSRAM controllers to the AHB bus matrix.

6.1 Features

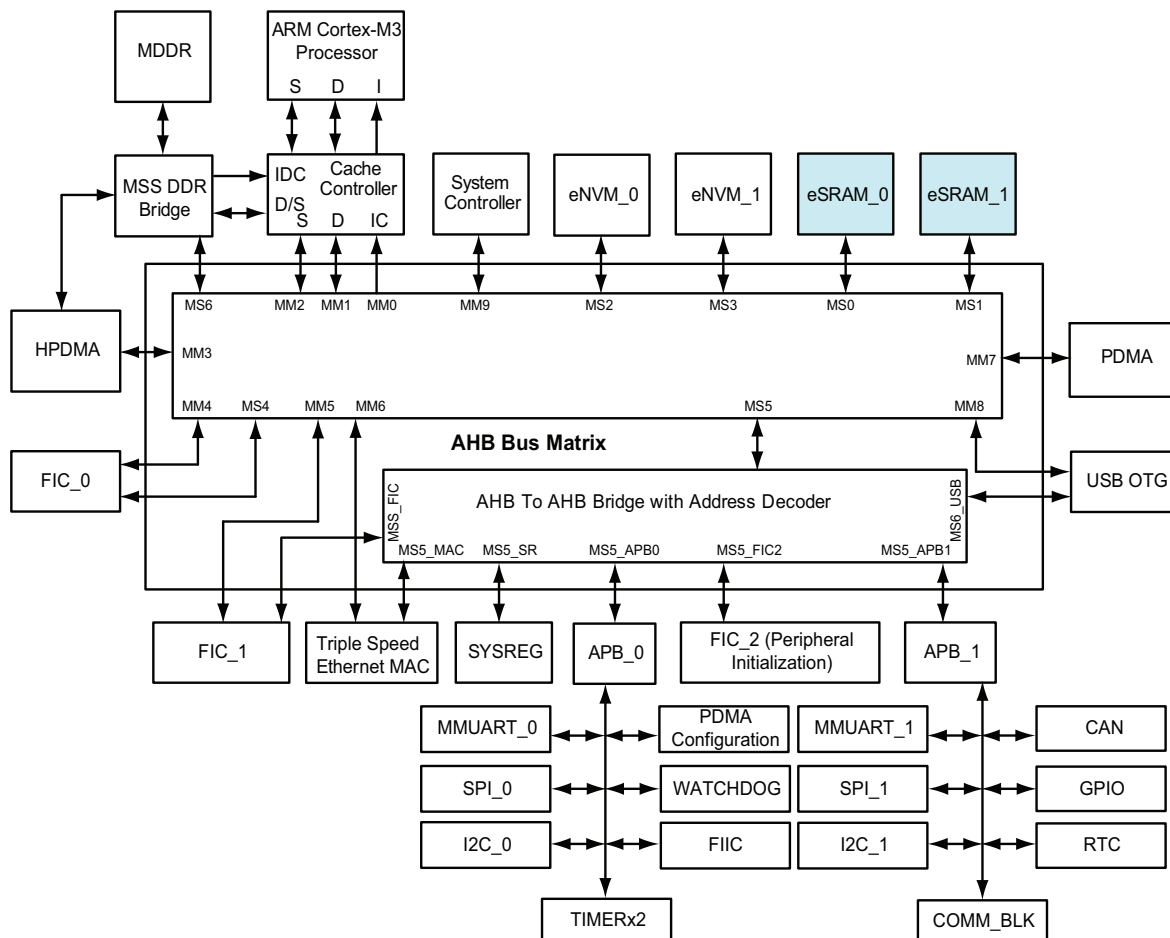
- Each eSRAM controller supports single bit error correction and dual bit error detection (SECDED).
- Two modes of operation: SECDED-ON and SECDED-OFF.
- The total amount of available eSRAM in each device is 64 KB in SECDED-ON mode and 80 KB in SECDED-OFF mode.
- Each individual eSRAM block is 32 KB in SECDED-ON mode and 40 KB in SECDED-OFF mode, organized in a $2 \times 4096 \times 40$ fashion.
- Having two blocks (eSRAM_0 and eSRAM_1) maximizes hardware parallelism. For example, at the same instant that the Cortex-M3 processor is reading from eSRAM_0, the Ethernet controller can read from eSRAM_1 independently.
- The eSRAM address space is byte, half-word (16-bit), and word (32-bit) addressable.
- A pipeline is provided to address the latency issues at higher speeds of operation.

As shown in the following figure, the total available size of the eSRAM is divided into two equal-sized blocks: eSRAM_0 and eSRAM_1. eSRAM_0 and eSRAM_1 are connected to slave 0 and slave 1 on the AHB bus matrix through eSRAM controller 0 and eSRAM controller 1.

The eSRAM controller is designed to interface to an 8192×40 RAM, which is organized in a $2 \times 4096 \times 40$ fashion with five 8-bit byte lanes in total. The Cortex-M3 processor and other masters find the eSRAMs available as one contiguous area of memory.

The following figure depicts the connectivity of eSRAM_0 and eSRAM_1 to the AHB bus matrix.

Figure 95 • eSRAM_0 and eSRAM_1 Connection to AHB Bus Matrix



6.2 Functional Description

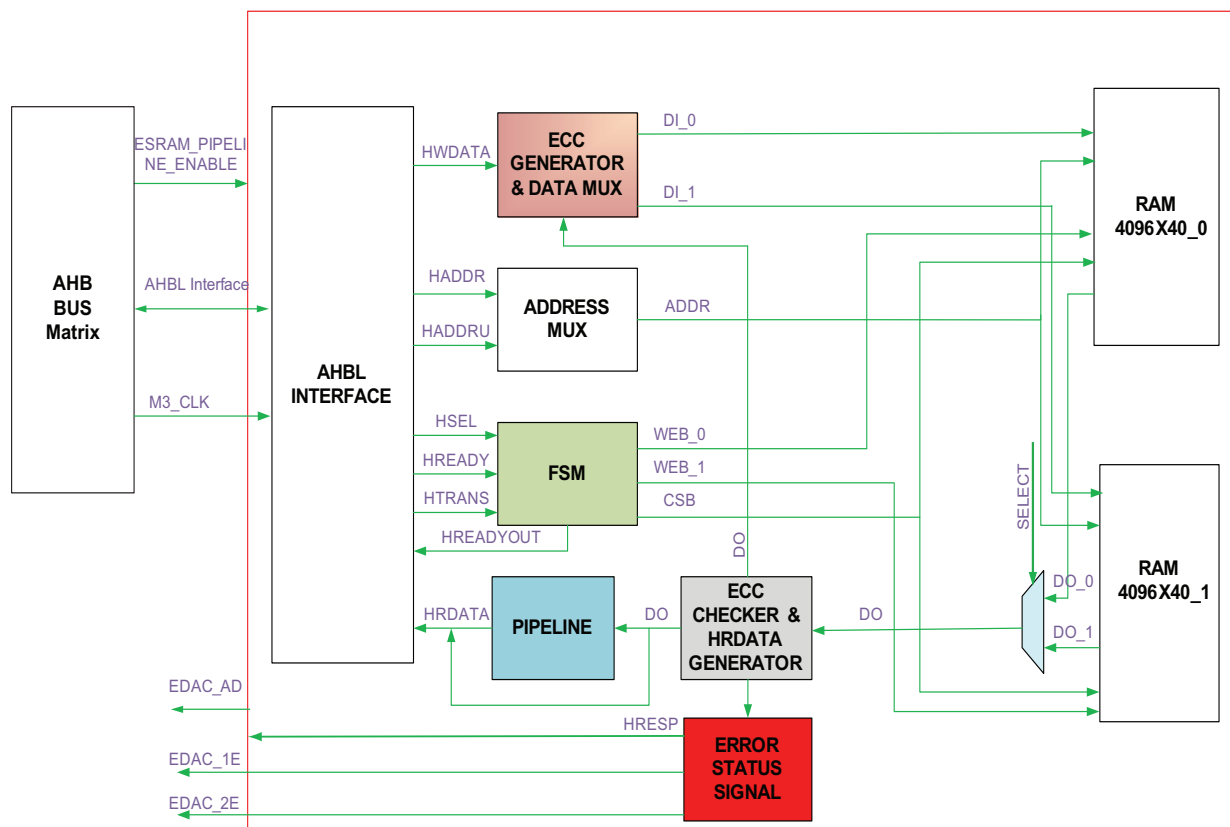
The following table lists the sizes of the eSRAM blocks and their address range.

Table 118 • eSRAM Block Sizes and Address Ranges

eSRAM Block	Physical RAM4096X40 Block	Size and Address Range with SECDED ON	Size and Address Range with SECDED OFF
eSRAM_0	RAM4096X40_0	16 KB from 0x20000000 to 0x20003FFF and ECC from 0x20010000 to 0x20010FFF	16 KB from 0x20000000 to 0x20003FFF and 4 KB from 0x20010000 to 0x20010FFF
	RAM4096X40_1	16 KB from 0x20004000 to 0x20007FFF and ECC from 0x20011000 to 0x20011FFF	16 KB from 0x20004000 to 0x20007FFF and 4 KB from 0x20011000 to 0x20011FFF
eSRAM_1	RAM4096X40_2	16 KB from 0x20008000 to 0x2000BFFF and ECC from 0x20012000 to 0x20012FFF	16 KB from 0x20008000 to 0x2000BFFF and 4 KB from 0x20012000 to 0x20012FFF
	RAM4096X40_3	16 KB from 0x2000C000 to 0x2000FFFF and ECC from 0x20013000 to 0x20013FFF	16 KB from 0x2000C000 to 0x2000FFFF and 4 KB from 0x20013000 to 0x20013FFF

The following figure shows the eSRAM controller blocks and their connectivity in SmartFusion2 FPGAs. Both eSRAMs and eSRAM controllers are identical in all design aspects.

Figure 96 • eSRAM Controller Block Diagram



M3_CLK is used within the MSS to clock the AHB bus matrix. Refer to [UG0449: SmartFusion2 and IGL002 Clocking Resources User Guide](#) for more information on M3_CLK.

AHBL Interface: Each eSRAM controller is an AHB-Lite (AHBL) slave that provides access to the eSRAM block from the AHB bus matrix.

ECC Generator and Data MUX: In SECEDED-ON mode, the ECC Generator generates the check bits for 32-bit data. For a 32-bit write from the AHBL interface, the input data AHB write data bus (HWDATA) is used to generate check bits. These check bits are appended to HWDATA and written to the memory. For 8-bit and 16-bit writes from the AHBL interface, a read-modify-write operation is used. This reads data from the 32-bit word, corrects if necessary, and then writes the new data value and ECC check bits.

In SECEDED-OFF mode, if the memory access is within 32 KB memory, HWDATA is sent directly to the memory input. If the access is for additional 8 KB memory, then the address for a particular byte of HWDATA will be selected based on the shift address.

Address MUX: This utilizes the AHB address bus (HADDR) and HADDRU (an additional HADDR bit) for selecting upper 8 K bank of the RAM. Based on the FSM internal signals, output ADDR is generated and passed to the memory. The shifted address is also generated and used for multiplexing data.

FSM: This generates output signal HREADYOUT and internal signals that are used for multiplexing an address.

Pipeline: A pipeline stage in the read path of eSRAM and the master that accesses this path is configurable using the ESRAM_PIPELINE_ENABLE signal. When ESRAM_PIPELINE_ENABLE is High, there is an extra one clock cycle delay for the read operation to maximize operational frequency. At higher frequencies (> 100 MHz) of Cortex-M3 or other masters accessing eSRAM, the eSRAM operations need an extra clock cycle for the correct data transactions.

ECC Checker and AHB Read Data Bus (HRDATA) Generator: In SECDED-ON mode, the ECC Checker takes data (DO) from the memory as the input during the read or read-modify-write cycle and checks for errors. One-bit errors detected are corrected.

If errors of more than one bit are detected, they are not corrected. In SECDED-OFF mode, the read out data is directly given as output from this block. Error Status Signals are set if any errors are detected.

Error Status Signals: Error bits are inputs from the ECC Checker. If one error bit is High, it causes the EDAC_1E signal to be High. In this case, there is no HRESP as the error is corrected. If there are two-bit errors, it cause the EDAC_2E signal to be High. In this case, HRESP is set High because the error is not corrected. The EDAC_1E and EDAC_2E signals are used to increment the ECC error counters within the SYSREG block (and the failing address is also passed to the SYSREG block). When the HRESET to ESRAMTOAHB is applied, it resets the EDAC address register which is maintained in ESRAMTOAHB and it does not clear the contents of SRAM. EDAC error counters are maintained in System Register which can be cleared either through same HRESET or by setting the [CLR_EDAC_COUNTERS](#).

6.2.1 Memory Organization

The 40 KB of eSRAM memory is divided into two banks: 32 KB and 8 KB, to store 32 bits of data and 7 check bits in SECDED-ON mode. Physically, however, the memory is organized as 4096×40 , which is 4096×5 bytes. When ECC is enabled, the fifth byte stores ECC values for the 32 bits of data. When ECC is disabled, the fifth byte location is used to create an additional 2 KB of user memory. Four locations are used for each 32-bit word.

The following table shows the organization of 4096×40 bits in SECDED-ON mode. The total size of the SRAM in the table is 40 KB. The locations show the memory used for the 32 KB block. ECC represents the 7-bit ECC.

Table 119 • SRAM Organization in SECDED-ON Mode

RAM 4096X40_1 4096 x 40 Bits						RAM 4096X40_0 4096 x 40 Bits				
Location	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
0	ECC	4003	4002	4001	4000	ECC	0003	0002	0001	0000
1	ECC	4007	4006	4005	4004	ECC	0007	0006	0005	0004
2046	ECC	5FFB	5FFA	5FF9	5FF8	ECC	1FFB	1FFA	1FF9	1FF8
2047	ECC	5FFF	5FFE	5FFD	5FFC	ECC	1FFF	1FFE	1FFD	1FFC
2048	ECC	6003	6002	6001	6000	ECC	2003	2002	2001	2000
2049	ECC	6007	6006	6005	6004	ECC	2007	2006	2005	2004
4094	ECC	7FFB	7FFA	7FF9	7FF8	ECC	3FFB	3FFA	3FF9	3FF8
4095	ECC	7FFF	7FFE	7FFD	7FFC	ECC	3FFF	3FFE	3FFD	3FFC

The following table shows the organization of 4096×40 bits in SECDED-OFF mode. The total size of the SRAM in the table is 40 KB. The red locations show the memory used for the 32 KB block. The green locations show memory used for the upper 8 KB block.

Table 120 • SRAM Organization in SECDED-OFF Mode

RAM 4096X40_1 4096 x 40 Bits						RAM 4096X40_0 4096 x 40 Bits				
Location	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
0	0001	4003	4002	4001	4000	0000	0003	0002	0001	0000
1	0005	4007	4006	4005	4004	0004	0007	0006	0005	0004
2046	1FF9	5FFB	5FFA	5FF9	5FF8	1FF8	1FFB	1FFA	1FF9	1FF8

Table 120 • SRAM Organization in SECDED-OFF Mode (continued)

2047	1FFD	5FFF	5FFE	5FFD	5FFC	1FFC	1FFF	1FFE	1FFD	1FFC
2048	0003	6003	6002	6001	6000	0002	2003	2002	2001	2000
2049	0007	6007	6006	6005	6004	0006	2007	2006	2005	2004
4094	1FFB	7FFB	7FFA	7FF9	7FF8	1FFA	3FFB	3FFA	3FF9	3FF8
4095	1FFF	7FFF	7FFE	7FFD	7FFC	1FFE	3FFF	3FFE	3FFD	3FFC

6.2.2 Modes of Operation

There are two modes of operation for the eSRAM controller: SECDED-ON and SECDED-OFF.

6.2.2.1 SECDED-ON

SECDED mode can be turned ON by configuring the [EDAC_CR](#) register. The total available memory for each eSRAM in this mode is 32 KB. The eSRAM controller generates 7 check bits for every 32 bits of data, so for every 32 bits of data there will be 7 bits of encoded data. The 7 bits of ECC allow 1-bit correction and 2-bit detection on the user data and ECC field. The 32 data bits and 7 bits of ECC are written to the memory with zero wait states. Byte and half-word write operations are done using a read-modify-write operation. The read-modify-write operation requires an additional wait state for byte and half-word write operations.

In the case of a 1-bit error, the previous 32 bits of data and ECC value are read and correction takes place automatically. The complete 32 bits plus ECC is rewritten. For byte and half-word write operations, there is one wait state required as the ECC value is read and corrected for the byte/half-word.

When a 2-bit error is detected during a read cycle for 32-bit data, HRESP is asserted High for two clock cycles and at the same time HREADYOUT goes Low for one clock cycle to indicate an error.

When a 2-bit error is detected during the read part of a read-modify-write byte or half-word operation, HRESP is asserted High.

6.2.2.2 SECDED-OFF

SECDED mode can be turned OFF by configuring the [EDAC_CR](#) register. The total available memory for each eSRAM is 40 KB. 1-bit correction and 2-bit detection on the user data is not applicable in this mode.

6.2.3 Pipeline Modes and Wait States for Read and Write Operations

When any master on the AHB bus matrix operates at a high frequency greater than 100 MHz, and is accessing eSRAM, an extra clock cycle is needed for transactions. An optional pipeline can be enabled on the Read data bus; this adds a clock cycle to all read operations. The pipeline is enabled by default in both SECDED-ON and SECDED-OFF modes. When the master on the AHB bus matrix operates at low frequency, less than 100 MHz, the pipeline can be turned off. Refer to [Table 126](#), page 201 for information on pipeline enable/ disable.

The actual frequency at which this is possible is specified in the AC characteristics table of [DS0451: IGLOO2 and SmartFusion2 Datasheet](#). When the pipeline is disabled, the number of wait states is less, increasing throughput of read operations.

The following table describes the wait states in different operation modes. These values indicate the number of wait states inserted by eSRAM controllers and apply to the reads and writes from masters within the MSS. Accessing eSRAM blocks from the FPGA fabric is performed through the fabric interface controller (FIC) interfaces. The FIC interface supports Bypass mode and Pipeline mode.

In Pipeline mode, the FIC interface adds one extra clock cycle for read and write, so the overall latency for accessing the eSRAM increases in this case.

Table 121 • Wait States in Different Operation Modes

Pipeline	eSRAM	SECEDED Mode	Operation	Size	Number of Wait States	Number of Wait States (Reads following a Write)
Enabled	32 KB RAM	SECEDED-ON Mode	Write	32-Bit	0	1
				16-Bit	1	3
				8-Bit	1	3
			Read	32-Bit	1	2
				16-Bit	1	2
				8-Bit	1	2
		SECEDED-OFF Mode	Write	32-Bit	0	0
				16-Bit	0	0
				8-Bit	0	0
			Read	32-Bit	1	2
				16-Bit	1	2
				8-Bit	1	2
	8 KB RAM	SECEDED-OFF Mode	Write	32-Bit	1	1
				16-Bit	0	0
				8-Bit	0	0
			Read	32-Bit	2	3
				16-Bit	1	2
				8-Bit	1	2

Table 121 • Wait States in Different Operation Modes (continued)

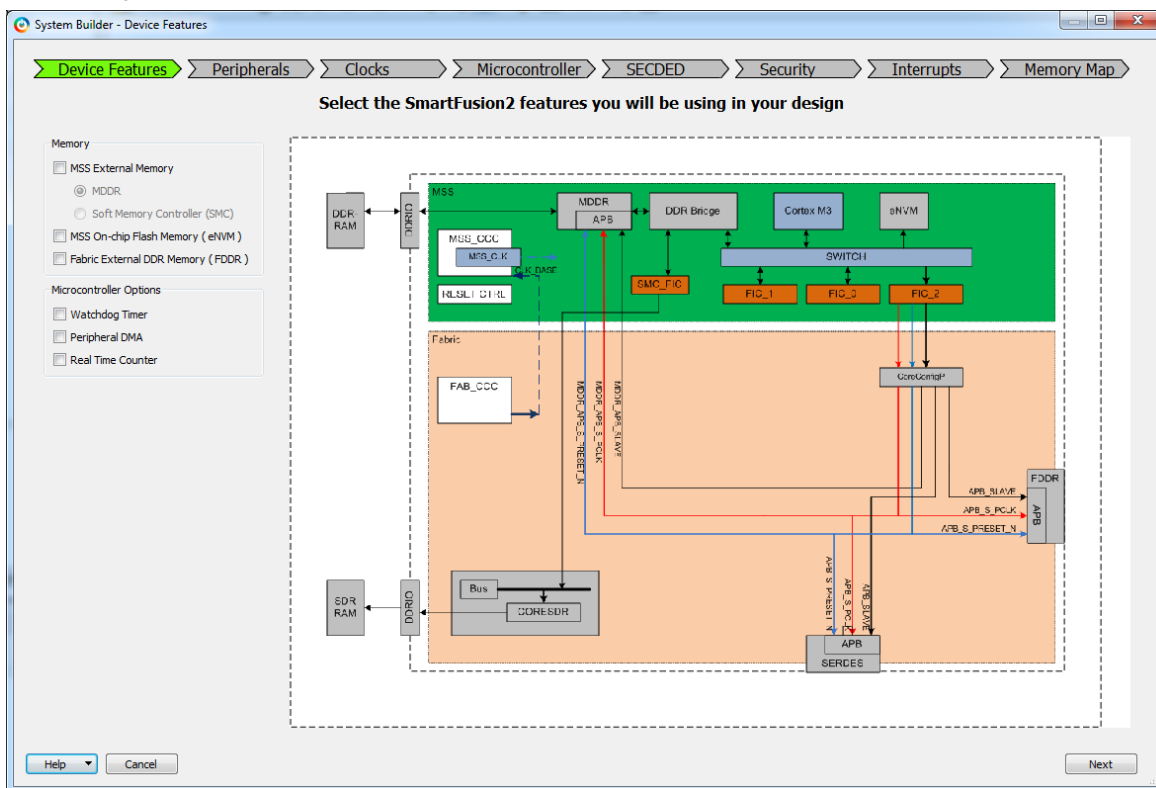
Pipeline	eSRAM	SECEDED Mode	Operation	Size	Number of Wait States	Number of Wait States (Reads following a Write)
Disabled	32 KB RAM	SECEDED-ON Mode	Write	32-Bit	0	0
				16-Bit	1	3
				8-Bit	1	3
			Read	32-Bit	0	1
				16-Bit	0	1
				8-Bit	0	1
		SECEDED-OFF Mode	Write	32-Bit	0	0
				16-Bit	0	0
				8-Bit	0	0
			Read	32-Bit	0	1
				16-Bit	0	1
				8-Bit	0	1
	8 KB RAM	SECEDED-OFF Mode	Write	32-Bit	1	1
				16-Bit	0	0
				8-Bit	0	0
			Read	32-Bit	1	2
				16-Bit	0	1
				8-Bit	0	1

6.3 How to Use eSRAM

This section describes how to use the eSRAM in the SmartFusion2 devices. To configure the SmartFusion2 device features and then build a complete system, use the **System Builder** graphical design wizard in the Libero SoC software.

The following figure shows the initial **System Builder** window where the required device features can be selected. For details on how to launch the **System Builder** wizard and a detailed information on how to use it, refer to the *SmartFusion2 System Builder User Guide*.

Figure 97 • System Builder Window

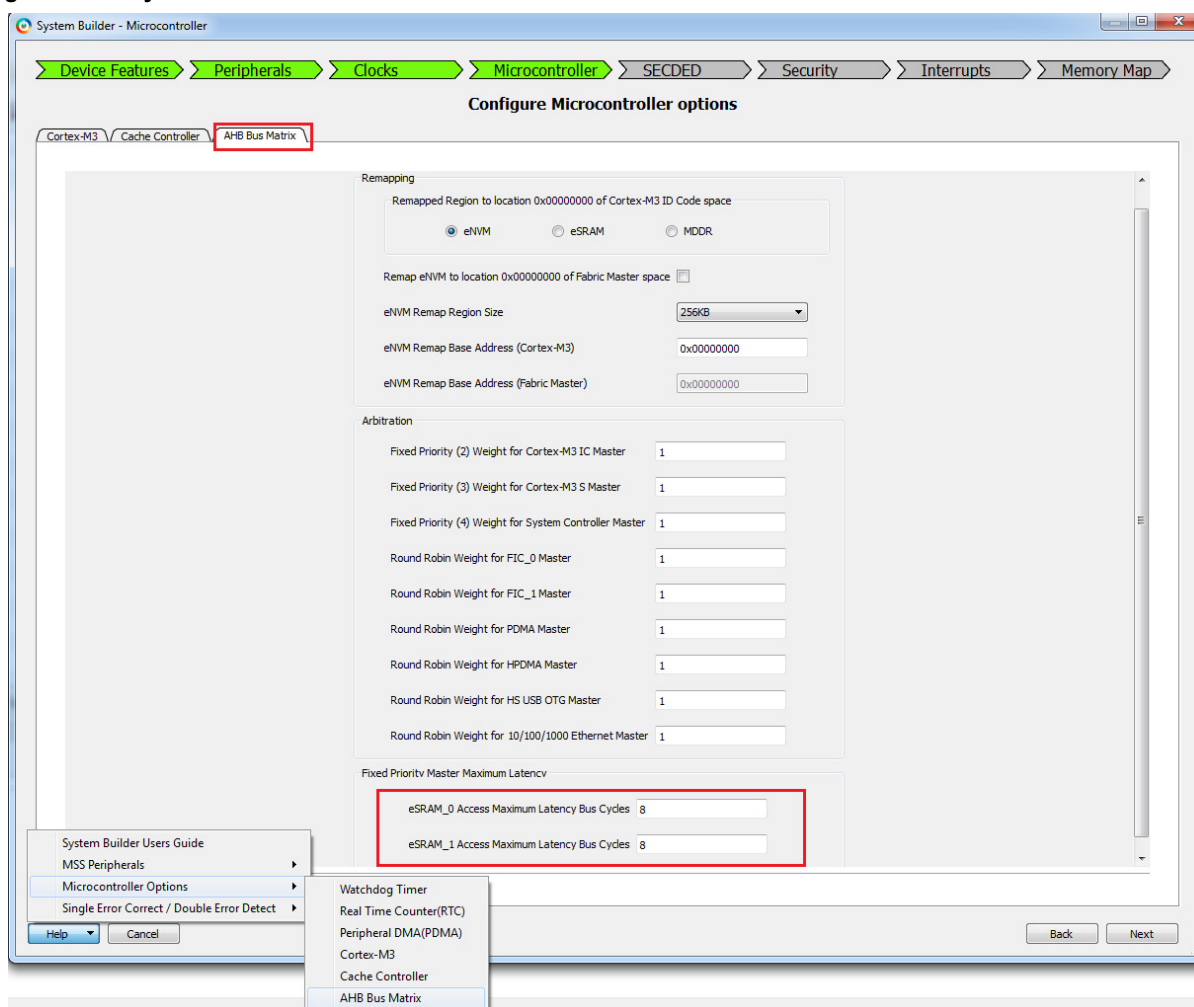


Any master (for example, Cortex-M3 processor, FPGA fabric master, HPDMA, or PDMA) connected to the AHB bus matrix can access the eSRAM blocks using the address range provided in [Table 118](#), page 188 for read and write operations.

The following steps are used to configure Programmable Slave Maximum Latency for eSRAM_0 and eSRAM_1 blocks and eSRAM SECCDED feature in the application using System Builder.

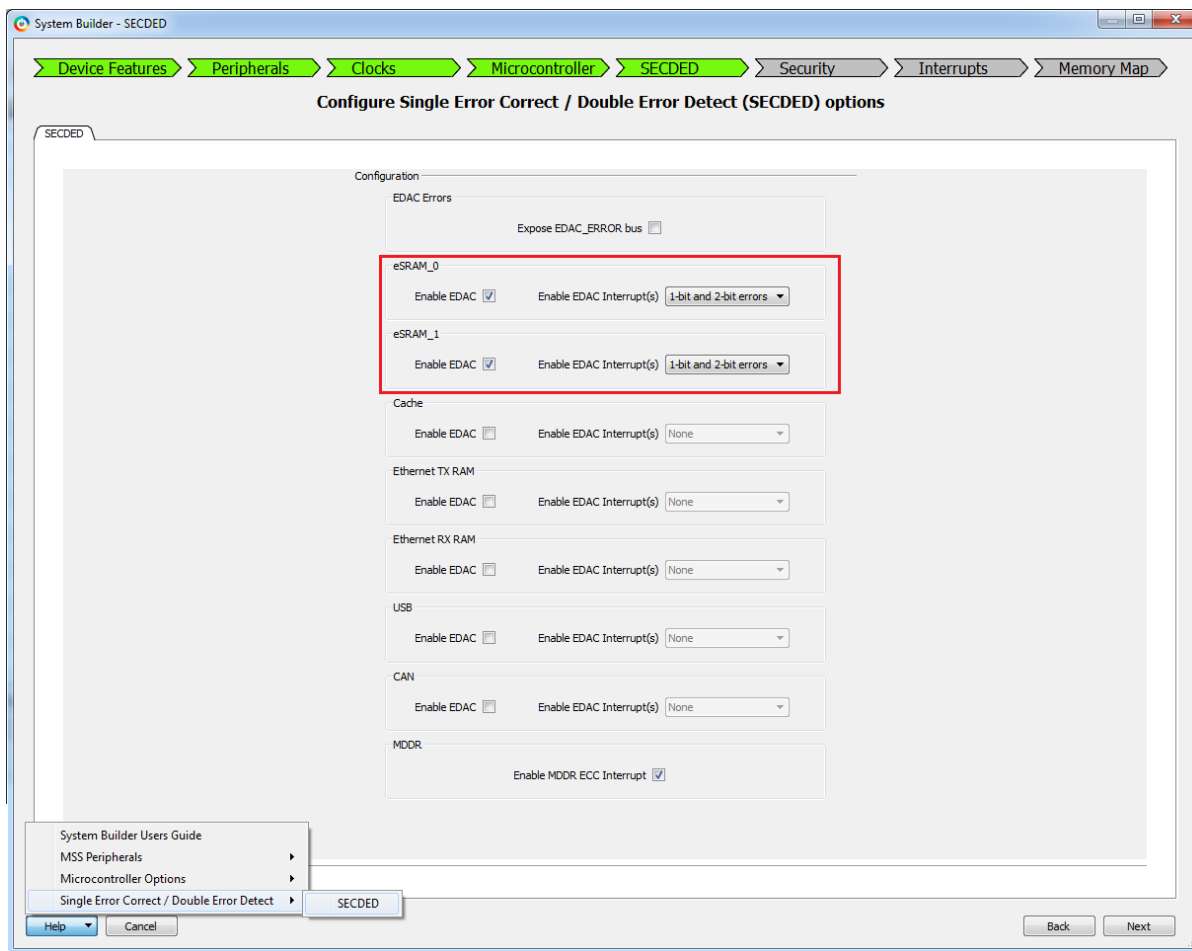
1. Navigate to the **Microcontroller** tab in the **System Builder** window to configure Programmable Slave Maximum Latency for eSRAM_0 and eSRAM_1 blocks. The following figure shows the **System Builder - Microcontroller** tab. For more information on the Programmable Slave Maximum Latency configuration and remapping eSRAM to Cortex-M3 code space, click **Help** and select AHB Bus Matrix document as shown in the figure. Refer to [AC390: SmartFusion2 SoC FPGA – Remapping eNVM, eSRAM, and DDR/SDR SDRAM Memories Application Note](#).

Figure 98 • System Builder - Microcontroller Tab



- Navigate to the **SECDED** tab in the **System Builder** window to configure SECDED options for eSRAM_0 and eSRAM_1. The following figure shows the **System Builder - SECDED** tab. For more information on SECDED, click **Help** and select SECDED document. Refer to [UG0388: SmartFusion2 SoC FPGA Error Detection and Correction of eSRAM Memory Demo User Guide](#).

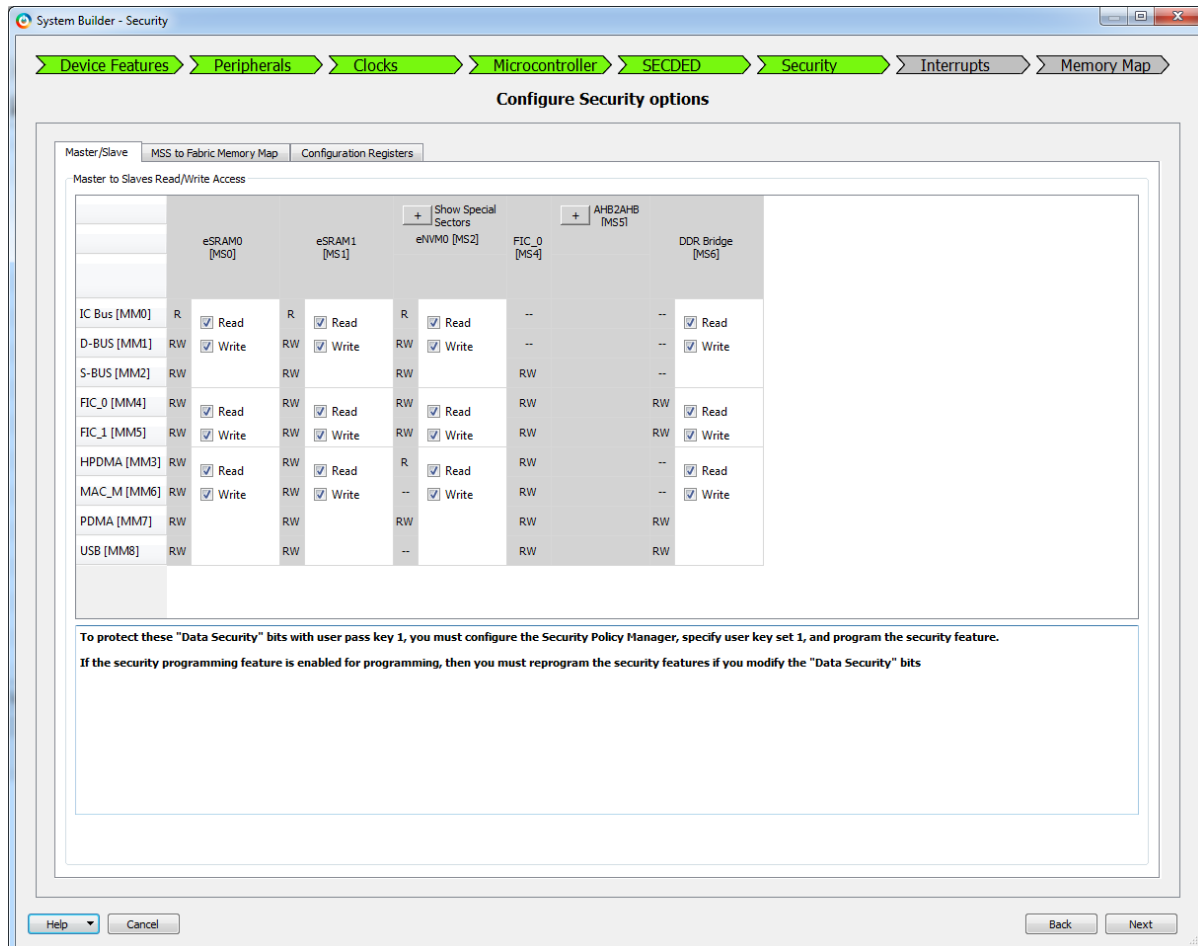
Figure 99 • System Builder - SECDED Tab



The SECDED feature can be enabled or disabled by selecting **Enable EDAC** for eSRAM_0 and eSRAM_1. The interrupts for 1-bit error or 2-bit error, or both 1-bit and 2-bit errors can be enabled.

- Navigate to the **Security** tab to select the read and write access permissions of eSRAM for different masters as shown in the following figure. The read and write permission for different masters are available for data and design security enabled devices like M2S050TS only. For more information on configuring the security options, refer to [SmartFusion2 MSS Security Configuration](#).

Figure 100 • System Builder - Security Tab



- Navigate to the **Memory Map** tab giving the required data in the rest of the **System Builder** tabs and click **Finish** to proceed with creating the MSS Subsystem.
- Do required Pin connections and Save the project. Generate the SmartDesign in Libero by clicking **Generate Component**.
- Double-click **Run PROGRAM Action** in the Libero **Design Flow** window to program the SmartFusion2 device.

Note: Refer to the **Accessing eNVM and eSRAM from the FPGA Fabric Logic Application Note** (to be released) for information on how to access the eSRAM using FPGA fabric logic.

6.4 SYSREG Control Registers

The registers listed in the following table control the behavior of the eSRAM. These registers are detailed in [SYSREG](#) and are listed here for clarity. Refer to the [System Register Block](#), page 670 for a detailed description of each register and bit.

Table 122 • SYSREG Control Registers

Register Name	Register Type	Flash Write Protect	Reset Source	Description
ESRAM_CR (0x40038000)	RW-P	Register	SYSRESET_N	Controls address mapping of the eSRAMs.
ESRAM_MAX_LAT (0x40038004)	RW-P	Register	SYSRESET_N	Configuration of maximum latency for accessing eSRAM_0 and eSRAM_1 slaves. This register gets updated by flash bit configuration set during device programming. This configuration can be done through the System Builder also using settings on the Microcontroller Tab .
ESRAM_PIPELINE_CR (0x40038080)	RW-P	Register	SYSRESET_N	Controls the pipeline present in the memory read path of eSRAM memory.
ESRAM0_EDAC_CNT (0x400380F0)	RO	N/A	SYSRESET_N	Represents 1-bit error count of eSRAM_0.
ESRAM1_EDAC_CNT (0x400380F4)	RO	N/A	SYSRESET_N	Represents 1-bit error count of eSRAM_1.
ESRAM0_EDAC_ADR (0x4003810C)	RO	N/A	SYSRESET_N	Address from eSRAM_0 on which 1-bit ECC error has occurred.
ESRAM1_EDAC_ADR (0x40038110)	RO	N/A	SYSRESET_N	Address from eSRAM_1 on which 1-bit ECC error has occurred.
MM0_1_2_SECURITY (0x40038124)	RO-U	N/A	SYSRESET_N	Read and Write security for Mirrored Master (MM) 0, 1, and 2 to eSRAM_0 and eSRAM_1.
MM4_5_DDR_FIC_SECURITY/MM4_5_FIC64_SECURITY (0x40038128)	RO-U	N/A	SYSRESET_N	Read and Write security for Mirrored Master (MM) 4, 5, and DDR_FIC to eSRAM_0 and eSRAM_1. This register gets updated by flash bit configuration set during device programming. This configuration can be done through the System Builder using settings on the Security tab.
MM3_6_7_8_SECURITY (0x4003812C)	RO-U	N/A	SYSRESET_N	Read and Write security for Mirrored Master (MM) 3, 6, 7, and 8 to eSRAM_0 and eSRAM_1. This register gets updated by flash bit configuration set during device programming. This configuration can be done through the System Builder using settings on the Security tab.

Table 122 • SYSREG Control Registers (continued)

Register Name	Register Type	Flash Write Protect	Reset Source	Description
MM9_SECURITY (0x40038130)	RO-U	N/A	SYSRESET_N	Read and Write security for Mirrored Master (MM) 9 to eSRAM_0 and eSRAM_1. This register gets updated by flash bit configuration set during device programming. This configuration can be done through the System Builder using settings on the Security tab.
EDAC_SR (0x40038190)	SW1C	N/A	SYSRESET_N	Status of 1-bit ECC error detection and correction (EDAC), 2-bit ECC error detection for eSRAM_0 and eSRAM_1. Individual register bits are set (1) when related input is asserted. Bits are individually cleared when corresponding register bit is written High.
CLR_EDAC_COUNTERS (0x400381A4)	W1P	N/A	SYSRESET_N	This is used to clear the 16-bit counter value in eSRAM_0 and eSRAM_1 corresponding to the count value of EDAC 1-bit and 2-bit errors.
EDAC_IRQ_ENABLE_CR (0x40038078)	RW-P	Register	SYSRESET_N	Enable/disable of 1-bit error, 2-bit error status update for eSRAM_0 and eSRAM_1. This can be set by the System Builder also using settings on the SECDED tab.
EDAC_CR (0x40038038)	RW-P	Register	SYSRESET_N	EDAC enable/disable and soft reset for eSRAM_0 and eSRAM_1. This can be set by the System Builder also using settings on the SECDED tab.

Table 123 • ESRAM_CR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	Reserved
1	SW_CC_ESRAM1FWREMAP	0	Defines the locations of eSRAM_0 and eSRAM_1 if eSRAM remap is enabled (if SW_CC_ESRAMFWREMAP is asserted). If SW_CC_ESRAMFWREMAP is 0, this bit has no meaning. If SW_CC_ESRAMFWREMAP is 1, this bit has the following definition: 0: eSRAM_0 is located at address 0x00000000 in the ICODE/DCODE space of Cortex-M3 processor and eSRAM_1 is located just above eSRAM_0 (adjacent to it). 1: eSRAM_1 is located at address 0x00000000 in ICODE/DCODE space of Cortex-M3 processor and eSRAM_0 is located just above eSRAM_1 (adjacent to it).

Table 123 • ESRAM_CR (continued)

0	SW_CC_ESRAMFWREMAP	0	<p>This bit indicates that eSRAM_0 and eSRAM_1 are remapped to ICODE/DCODE space of the Cortex-M3 processor. If this bit is 1 and SW_CC_ESRAM1FWREMAP is 0, then eSRAM_0 is at location 0x00000000 and eSRAM_1 is always remapped to be just above eSRAM_0 (the two eSRAMs are adjacent in ICODE/DCODE space). Both eSRAMs also remain visible in SYSTEM space of the Cortex-M3 processor and remain visible at this location to all other (non-Cortex-M3 processor) masters. The bit definitions:</p> <p>0: No eSRAM remap is enabled. This means that eNVM (or MDDR) is present at location 0x00000000.</p> <p>1: eSRAM_0 and eSRAM_1 are remapped to location 0x00000000 of Cortex-M3 processor ICODE/DCODE space.</p>
---	--------------------	---	---

Table 124 • ESRAM_MAX_LAT

Bit Number	Name	Reset Value	Description
[31:6]	Reserved	0	Reserved
[5:3]	SW_MAX_LAT_ESRAM1	0x1	Defines the maximum number of cycles the processor bus will wait for eSRAM1 when it is being accessed by a master with a weighted round robin (WRR) priority scheme. The latency values are as given in Table 125 , page 200.
[2:0]	SW_MAX_LAT_ESRAM0	0x1	Defines the maximum number of cycles the processor bus will wait for eSRAM0 when it is being accessed by a master with a WRR priority scheme. It is configurable from 1 to 8 (8 by default). The latency values are as given in Table 125 , page 200.

The following table gives eSRAM maximum latency values, where x is either 0 or 1.

Table 125 • eSRAM Maximum Latency Values

SW_MAX_LAT_ESRAM<X>	Latency
0	8 (default)
1	1
2	2
3	3
4	4
5	5
6	6
7	7

Table 126 • ESRAM_PIPELINE_CR

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	Reserved
0	ESRAM_PIPELINE_ENABLE	0x1	Controls the pipeline present in the read path of eSRAM memory. Allowed values: 0: Pipeline will be bypassed. 1: Pipeline will be present in the memory read path.

Table 127 • ESRAM0_EDAC_CNT

Bit Number	Name	Reset Value	Description
[31:16]	ESRAM0_EDAC_CNT_2E	0	16-bit counter that counts the number of 2-bit uncorrected errors for eSRAM0. The counter will not roll back and will stay at its maximum value.
[15:0]	ESRAM0_EDAC_CNT_1E	0	16-bit counter that counts the number of 1-bit corrected errors for eSRAM0. The counter will not roll back and will stay at its maximum value.

Note: Refer to [Table 136](#), page 206 to clear the counter.

Table 128 • ESRAM1_EDAC_CNT

Bit Number	Name	Reset Value	Description
[31:16]	ESRAM1_EDAC_CNT_2E	0	16-bit counter that counts the number of 2-bit uncorrected errors for eSRAM1. The counter will not roll back and will stay at its maximum value.
[15:0]	ESRAM1_EDAC_CNT_1E	0	16-bit counter that counts the number of 1-bit corrected errors for eSRAM1. The counter will not roll back and will stay at its maximum value.

Note: Refer to [Table 136](#), page 206 to clear the counter.

Table 129 • ESRAM0_EDAC_ADR

Bit Number	Name	Reset Value	Description
[31:25]	Reserved	0	Reserved
[25:13]	ESRAM0_EDAC_2E_AD	0	Stores the address from eSRAM0 on which a 2-bit SECCDED error has occurred.
[12:0]	ESRAM0_EDAC_1E_AD	0	Stores the address from eSRAM0 on which a 1-bit SECCDED error has occurred.

Table 130 • ESRAM1_EDAC_ADR

Bit Number	Name	Reset Value	Description
[31:25]	Reserved	0	Reserved
[25:13]	ESRAM1_EDAC_2E_AD	0	Stores the address from eSRAM1 on which a 2-bit SECEDED error has occurred.
[12:0]	ESRAM1_EDAC_1E_AD	0	Stores the address from eSRAM1 on which a 1-bit SECEDED error has occurred.

Table 131 • MM0_1_2_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	Reserved
9	MM0_1_2_MS6_ALLOWED_W	1	Write security bits for Masters 0, 1, and 2 to Slave 6 (MSS DDR bridge). If not set, Masters 0, 1, and 2 will not have write access to Slave 6.
8	MM0_1_2_MS6_ALLOWED_R	1	Read security bits for Masters 0, 1, and 2 to Slave 6 (MSS DDR bridge). If not set, Masters 0, 1, and 2 will not have read access to Slave 6.
7	MM0_1_2_MS3_ALLOWED_W	1	Write security bits for Masters 0, 1, and 2 to Slave 3 (eNVM1). If not set, Masters 0, 1, and 2 will not have write access to Slave 3.
6	MM0_1_2_MS3_ALLOWED_R	1	Read security bits for Masters 0, 1, and 2 to Slave 3 (eNVM1). If not set, Masters 0, 1, and 2 will not have read access to Slave 3.
5	MM0_1_2_MS2_ALLOWED_W	1	Write security bits for Masters 0, 1, and 2 to Slave 2 (eNVM0). If not set, Masters 0, 1, and 2 will not have write access to Slave 2.
4	MM0_1_2_MS2_ALLOWED_R	1	Read security bits for Masters 0, 1, and 2 to Slave 2 (eNVM0). If not set, Masters 0, 1, and 2 will not have read access to Slave 2.
3	MM0_1_2_MS1_ALLOWED_W	1	Write security bits for Masters 0, 1, and 2 to Slave 1 (eSRAM1). If not set, Masters 0, 1, and 2 will not have write access to Slave 1.
2	MM0_1_2_MS1_ALLOWED_R	1	Read security bits for Masters 0, 1, and 2 to Slave 1 (eSRAM1). If not set, Masters 0, 1, and 2 will not have read access to Slave 1.
1	MM0_1_2_MS0_ALLOWED_W	1	Write security bits for Masters 0, 1, and 2 to Slave 0 (eSRAM0). If not set, Masters 0, 1, and 2 will not have write access to Slave 0.
0	MM0_1_2_MS0_ALLOWED_R	1	Read security bits for Masters 0, 1, and 2 to Slave 0 (eSRAM0). If not set, Masters 0, 1, and 2 will not have read access to Slave 0.

Note: Refer to [Figure 101](#), page 210 for more information on AHB Bus Matrix masters and slaves.

Table 132 • MM4_5_DDR_FIC_SECURITY/MM4_5_FIC64_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	Reserved
9	MM4_5_DDR_FIC_MS6_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 6 (MSS DDR bridge). If not set, masters 4, 5 and DDR_FIC will not have write access to slave 6.
8	MM4_5_DDR_FIC_MS6_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 6 (MSS DDR bridge). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 6.
7	MM4_5_DDR_FIC_MS3_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 3 (eNVM1). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 3.
6	MM4_5_DDR_FIC_MS3_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 3 (eNVM1). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 3.
5	MM4_5_DDR_FIC_MS2_ALLOWED_W	1	Write Security Bits for masters 4, 5, and DDR_FIC to slave 2 (eNVM0). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 2.
4	MM4_5_DDR_FIC_MS2_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 2 (eNVM0). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 2.
3	MM4_5_DDR_FIC_MS1_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 1 (eSRAM1). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 1.
2	MM4_5_DDR_FIC_MS1_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 1 (eSRAM1). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 1.
1	MM4_5_DDR_FIC_MS0_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 0 (eSRAM0). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 0.
0	MM4_5_DDR_FIC_MS0_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 0 (eSRAM0). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 0.

Note: Refer to [Figure 101](#), page 210 for more information on AHB Bus Matrix masters and slaves.

Table 133 • MM3_6_7_8_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	Reserved
9	MM3_6_7_8_MS6_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 6 (MSS DDR bridge). If not set, masters 3, 6, 7, and 8 will not have write access to slave 6.
8	MM3_6_7_8_MS6_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 6 (MSS DDR bridge). If not set, masters 3, 6, 7, and 8 will not have read access to slave 6.
7	MM3_6_7_8_MS3_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 3 (eNVM1). If not set, masters 3, 6, 7, and 8 will not have write access to slave 3.
6	MM3_6_7_8_MS3_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 3 (eNVM1). If not set, masters 3, 6, 7, and 8 will not have read access to slave 3.
5	MM3_6_7_8_MS2_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 2 (eNVM0). If not set, masters 3, 6, 7, and 8 will not have write access to slave 2.
4	MM3_6_7_8_MS2_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 2 (eNVM0). If not set, masters 3, 6, 7, and 8 will not have read access to slave 2.
3	MM3_6_7_8_MS1_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 1 (eSRAM1). If not set, masters 3, 6, 7, and 8 will not have write access to slave 1.
2	MM3_6_7_8_MS1_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 1 (eSRAM1). If not set, masters 3, 6, 7, and 8 will not have read access to slave 1.
1	MM3_6_7_8_MS0_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 0 (eSRAM0). If not set, masters 3, 6, 7, and 8 will not have write access to slave 0.
0	MM3_6_7_8_MS0_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 0 (eSRAM0). If not set, masters 3, 6, 7, and 8 will not have read access to slave 0.

Note: Refer to [Figure 101](#), page 210 for more information on AHB Bus Matrix masters and slaves.

Table 134 • MM9_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	Reserved
9	MM9_MS6_ALLOWED_W	1	Write security bits for master 9 to slave 6 (MSS DDR bridge). If not set, master 9 will not have write access to slave 6.
8	MM9_MS6_ALLOWED_R	1	Read security bits for master 9 to slave 6 (MSS DDR bridge). If not set, master 9 will not have read access to slave 6.

Table 134 • MM9_SECURITY (continued)

Bit Number	Name	Reset Value	Description
7	MM9_MS3_ALLOWED_W	1	Write security bits for master 9 to slave 3 (eNVM1). If not set, master 9 will not have write access to slave 3.
6	MM9_MS3_ALLOWED_R	1	Read security bits for master 9 to slave 3 (eNVM1). If not set, master 9 will not have read access to slave 3.
5	MM9_MS2_ALLOWED_W	1	Write security bits for master 9 to slave 2 (eNVM0). If not set, master 9 will not have write access to slave 2.
4	MM9_MS2_ALLOWED_R	1	Read security bits for master 9 to slave 2 (eNVM0). If not set, master 9 will not have read access to slave 2.
3	MM9_MS1_ALLOWED_W	1	Write security bits for master 9 to slave 1 (eSRAM1). If not set, master 9 will not have write access to slave 1.
2	MM9_MS1_ALLOWED_R	1	Read security bits for master 9 to slave 1 (eSRAM1). If not set, master 9 will not have read access to slave 1.
1	MM9_MS0_ALLOWED_W	1	Write security bits for master 9 to slave 0 (eSRAM0). If not set, master 9 will not have write access to slave 0.
0	MM9_MS0_ALLOWED_R	1	Read security bits for master 9 to slave 0 (eSRAM0). If not set, master 9 will not have read access to slave 0.

Note: Refer to [Figure 101](#), page 210 for more information on AHB Bus Matrix masters and slaves.

Table 135 • EDAC_SR

Bit Number	Name	Reset Value	Description
[31:14]	Reserved	0	Reserved
13	CAN_EDAC_2E	0	Updated by CAN when a 2-bit SECEDED error has been detected for RAM memory.
12	CAN_EDAC_1E	0	Updated by CAN when a 1-bit SECEDED error has been detected and is corrected for RAM memory.
11	USB_EDAC_2E	0	Updated by USB when a 2-bit SECEDED error has been detected for RAM memory.
10	USB_EDAC_1E	0	Updated by USB when a 1-bit SECEDED error has been detected and is corrected for RAM memory.
9	MAC_EDAC_RX_2E	0	Updated by Ethernet when a 2-bit SECEDED error has been detected for Rx RAM memory.
8	MAC_EDAC_RX_1E	0	Updated by Ethernet when a 1-bit SECEDED error has been detected and is corrected for Rx RAM memory.
7	MAC_EDAC_TX_2E	0	Updated by Ethernet when a 2-bit SECEDED error has been detected for Tx RAM memory.
6	MAC_EDAC_TX_E	0	Updated by Ethernet when a 1-bit SECEDED error has been detected and is corrected for Tx RAM memory.
5	Reserved	0	Reserved
4	Reserved	0	Reserved

Table 135 • EDAC_SR (continued)

Bit Number	Name	Reset Value	Description
3	ESRAM1_EDAC_2E	0	Updated by the eSRAM_1 controller when a 2-bit SECEDED error has been detected for eSRAM1 memory.
2	ESRAM1_EDAC_1E	0	Updated by the eSRAM_1 Controller when a 1-bit SECEDED error has been detected and is corrected for eSRAM1 memory.
1	ESRAM0_EDAC_2E	0	Updated by the eSRAM_0 controller when a 2-bit SECEDED error has been detected for eSRAM0 memory.
0	ESRAM0_EDAC_1E	0	Updated by the eSRAM_0 controller when a 1-bit SECEDED error has been detected and is corrected for eSRAM0 memory.

Table 136 • CLR_EDAC_COUNTERS

Bit Number	Name	Reset Value	Description
[31:14]	Reserved	0	Reserved
13	CAN_EDAC_CNTCLR_2E	0	Generated to clear the 16-bit counter value in CAN corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the CAN_EDAC_CNT register.
12	CAN_EDAC_CNTCLR_1E	0	Generated to clear the 16-bit counter value in CAN corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the CAN_EDAC_CNT register.
11	USB_EDAC_CNTCLR_2E	0	Generated to clear the 16-bit counter value in USB corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the USB_EDAC_CNT register.
10	USB_EDAC_CNTCLR_1E	0	Generated to clear the 16-bit counter value in USB corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the USB_EDAC_CNT register.
9	MAC_EDAC_RX_CNTCLR_2E	0	Generated to clear the 16-bit counter value in Ethernet MAC Rx RAM corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the MAC_EDAC_RX_CNT register.
8	MAC_EDAC_RX_CNTCLR_1E	0	Generated to clear the 16-bit counter value in Ethernet MAC Rx RAM corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the MAC_EDAC_RX_CNT register.
7	MAC_EDAC_TX_CNTCLR_2E	0	Generated to clear the 16-bit counter value in Ethernet MAC Tx RAM corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the MAC_EDAC_TX_CNT register.
6	MAC_EDAC_TX_CNTCLR_1E	0	Generated to clear the 16-bit counter value in Ethernet MAC Tx RAM corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the MAC_EDAC_TX_CNT register.
5	Reserved	0	Reserved
4	Reserved	0	Reserved

Table 136 • CLR_EDAC_COUNTERS (continued)

Bit Number	Name	Reset Value	Description
3	ESRAM1_EDAC_CNTCLR_2E	0	Generated to clear the 16-bit counter value in ESRAM1 corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the ESRAM1_EDAC_CNT register.
2	ESRAM1_EDAC_CNTCLR_1E	0	Generated to clear the 16-bit counter value in eSRAM1 corresponding to count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the ESRAM1_EDAC_CNT register.
1	ESRAM0_EDAC_CNTCLR_2E	0	Generated to clear the 16-bit counter value in ESRAM0 corresponding to count value of EDAC 2bit Errors. This in turn clears the upper 16 bits of the ESRAM0_EDAC_CNT register.
0	ESRAM0_EDAC_CNTCLR_1E	0	Generated to clear the 16-bit counter value in ESRAM0 corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the ESRAM0_EDAC_CNT register.

Table 137 • EDAC_IRQ_ENABLE_CR

Bit Number	Name	Reset Value	Description
[31:15]	Reserved	0	Reserved
14	MDDR_ECC_INT_EN	0	Allows the error EDAC for MDDR status update to be disabled. Allowed values: 0: MDDR_EDAC_2E_EN is disabled. 1: MDDR_EDAC_2E_EN is enabled.
13	CAN_EDAC_2E_EN	0	Allows the 2-bit error EDAC for CAN status update to be disabled. Allowed values: 0: CAN_EDAC_2E_EN is disabled. 1: CAN_EDAC_2E_EN is enabled.
12	CAN_EDAC_1E_EN	0	Allows the 1-bit error EDAC for CAN status update to be disabled. Allowed values: 0: CAN_EDAC_1E_EN is disabled. 1: CAN_EDAC_1E_EN is enabled.
11	USB_EDAC_2E_EN	0	Allows the 2-bit error EDAC for USB status update to be disabled. Allowed values: 0: USB_EDAC_2E_EN is disabled. 1: USB_EDAC_2E_EN is enabled.
10	USB_EDAC_1E_EN	0	Allows the 1-bit error EDAC for USB status update to be disabled. Allowed values: 0: USB_EDAC_1E_EN is disabled. 1: USB_EDAC_1E_EN is enabled.
9	MAC_EDAC_RX_2E_EN	0	Allows the 2-bit error EDAC for Ethernet Rx RAM status update to be disabled. Allowed values: 0: MAC_EDAC_RX_2E_EN is disabled. 1: MAC_EDAC_RX_2E_EN is enabled.

Table 137 • EDAC_IRQ_ENABLE_CR (continued)

Bit Number	Name	Reset Value	Description
8	MAC_EDAC_RX_1E_EN	0	Allows the 1-bit error EDAC for Ethernet Rx RAM status update to be disabled. Allowed values: 0: MAC_EDAC_RX_1E_EN is disabled. 1: MAC_EDAC_RX_1E_EN is enabled.
7	MAC_EDAC_TX_2E_EN	0	Allows the 2-bit error EDAC for Ethernet Tx RAM status update to be disabled. Allowed values: 0: MAC_EDAC_TX_2E_EN is disabled. 1: MAC_EDAC_TX_2E_EN is enabled.
6	MAC_EDAC_TX_1E_EN	0	Allows the 1-bit error EDAC for Ethernet Tx RAM status update to be disabled. Allowed values: 0: MAC_EDAC_TX_1E_EN is disabled. 1: MAC_EDAC_TX_1E_EN is enabled.
5	Reserved	0	Reserved
4	Reserved	0	Reserved
3	ESRAM1_EDAC_2E_EN	0	Allows the 2-bit error EDAC for eSRAM1 status update to be disabled. Allowed values: 0: ESRAM1_EDAC_2E_EN is disabled. 1: ESRAM1_EDAC_2E_EN is enabled.
2	ESRAM1_EDAC_1E_EN	0	Allows the 1-bit error EDAC for eSRAM1 status update to be disabled. Allowed values: 0: ESRAM1_EDAC_1E_EN is disabled. 1: ESRAM1_EDAC_1E_EN is enabled.
1	ESRAM0_EDAC_2E_EN	0	Allows the 2-bit error EDAC for eSRAM0 status update to be disabled. Allowed values: 0: ESRAM0_EDAC_2E_EN is disabled. 1: ESRAM0_EDAC_2E_EN is enabled.
0	ESRAM0_EDAC_1E_EN	0	Allows the 1-bit error EDAC for eSRAM0 status update to be disabled. Allowed values: 0: ESRAM0_EDAC_1E_EN is disabled. 1: ESRAM0_EDAC_1E_EN is enabled.

Table 138 • EDAC_CR

Bit Number	Name	Reset Value	Description
[31:7]	Reserved	0	Reserved.
6	CAN_EDAC_EN	0	Allows the EDAC for CAN to be disabled. Allowed values: 0: EDAC disabled 1: EDAC enabled
5	USB_EDAC_EN	0	Allows the EDAC for USB to be disabled. Allowed values: 0: EDAC disabled 1: EDAC enabled
4	MAC_EDAC_RX_EN	0	Allows the EDAC for Ethernet Rx RAM to be disabled. Allowed values: 0: Rx RAM EDAC disabled 1: Rx RAM EDAC enabled

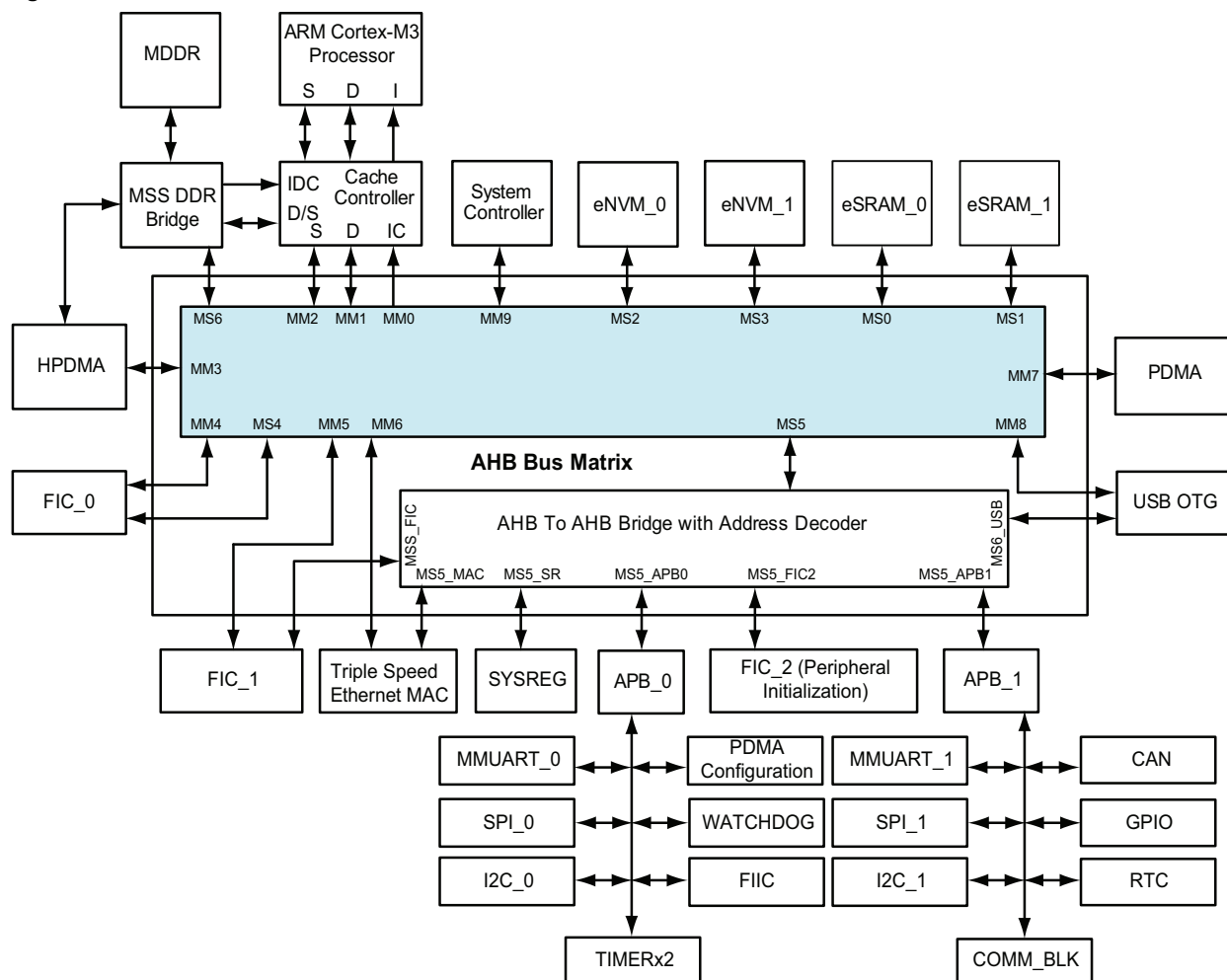
Table 138 • EDAC_CR (continued)

3	MAC_EDAC_TX_EN	0	Allows the EDAC for Ethernet Tx RAM to be disabled. Allowed values: 0: Tx RAM EDAC disabled 1: Tx RAM EDAC enabled
2	Reserved	0	Reserved
1	ESRAM1_EDAC_EN	0	Allows the EDAC for eSRAM1 to be disabled. Allowed values: 0: EDAC disabled 1: EDAC enabled
0	ESRAM0_EDAC_EN	0	Allows the EDAC for eSRAM0 to be disabled. Allowed values: 0: EDAC disabled 1: EDAC enabled

7 AHB Bus Matrix

The AHB bus matrix is a multi-layer AHB matrix. It is not a full crossbar switch, but a customized subset of a full switch. It works purely as an AHB-Lite matrix. The SmartFusion2 SoC FPGA AHB bus matrix has ten masters and seven direct slaves as depicted in the following figure. One master is permitted to access a slave at the same time another master is accessing a different slave. If more than one master is attempting to access the same slave simultaneously, arbitration for that slave is performed. Arbitration is not purely round robin or weighted round robin (WRR), but it is a combination of priority, round robin for processor-related masters and WRR for non-processor masters.

Figure 101 • AHB Bus Matrix Masters and Slaves



The preceding figure depicts the connectivity of masters and slaves in the AHB bus matrix. Nomenclature such as MM0 and MS0 refers to a mirrored master and a mirrored slave. A mirrored master port in the matrix connects directly to an AHB master; it has the same set of signals, but the direction of the signals is described relative to the other end of the connection.

A mirrored slave port in the matrix connects directly to an AHB slave. Only a subset of the full set of theoretical paths is implemented within the AHB bus matrix. The AHB bus matrix performs the address decoding of all slaves except for slaves that connect to the AHB-to-AHB bridge.

7.1 Functional Description

This section provides a detailed description of the AHB bus matrix.

7.1.1 Architecture Overview

Figure 102, page 212 depicts the interconnection between the master stage blocks and the slave stage blocks. The basic building blocks of the AHB bus matrix are the master stage block with an address decoder and the slave stage block with a slave arbiter. Each master interfaces with the master stage block and each slave interfaces with the slave stage block. The masters and slaves connect as shown in Figure 101, page 210.

An address decoder sub-block in each master stage generates the slave select signal to the corresponding slave. A slave arbiter sub-block in each slave stage generates the address-ready signal to the selected master.

Table 139 • AHB Bus Matrix Connectivity

	Masters									
	M3 DCode Bus	M3 ICode Bus	M3 System Bus	System Controller	HPDM A	FIC_0	FIC_1	MAC	PDMA	USB
	MM1	MM0	MM2	MM9	MM3	MM4	MM5	MM6	MM7	MM8
Priority	1	2	3	4	4	4	4	4	4	4
Arbitration	Fixed	Fixed	Fixed	Fixed	WRR	WRR	WRR	WRR	WRR	WRR
eSRAM0	MS0 RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
eSRAM1	MS1 RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
eNVM_0	MS2 RW ¹	R ¹	RW ¹	RW ¹	R ¹	RW ¹	RW ¹		RW ¹	
eNVM_1	MS3 RW ¹	R ¹	RW ¹	RW ¹	R ¹	RW ¹	RW ¹		RW ¹	
FIC_0	MS4		RW	RW	RW	RW	RW	RW	RW	RW
MAC	MS5		RW	RW		RW	RW			
FIC_1			RW	RW	RW	RW	RW	RW	RW	RW
SYSREG			RW	RW		RW	RW			
APB_0			RW	RW		RW	RW		RW	
APB_1			RW	RW		RW	RW		RW	
APB_2			RW	RW		RW	RW			
USB			RW	RW		RW	RW			
MSS DDR Bridge	MS6			RW		RW	RW	RW	RW	RW

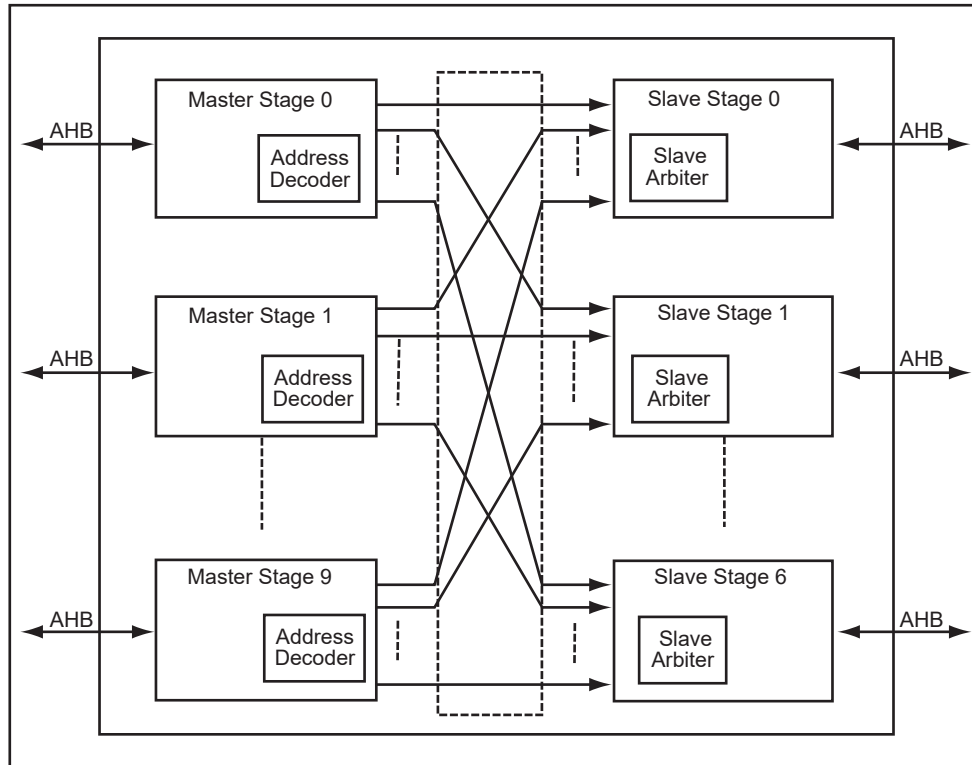
- Exercise caution when commanding the eNVM to program or erase data. Other masters in the system may not be aware that the eNVM is unavailable if it is in a program or erase cycle. Microsemi recommends you use some form of software semaphore to control access.
- Low numbers in priority represent higher priority, with 1 being the highest priority.

Reads or writes to areas not allowed cause the AHB bus matrix to complete the transaction with an HRESP error indication. An error bit is set in the SW_ERRORSTATUS field of the MSS_EXTERNAL_SR register. The following types of errors can occur:

- Write by an enabled master to a slave that is not RW
- Write by an enabled master to addresses not corresponding to a slave
- Write by the fabric master to the protected region
- Write by a disabled master to any location

- Read by an enabled master to any slave that is not R or RW
- Read by an enabled master to addresses not corresponding to a slave
- Read by the fabric master to the protected region
- Read by a disabled master to any location

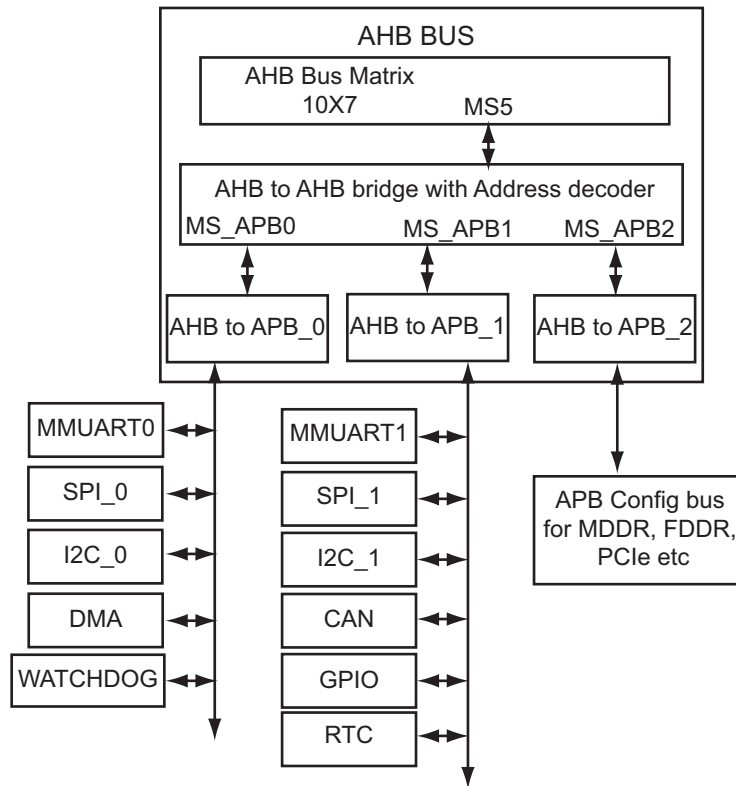
Figure 102 • Master Stage and Slave Stage Interconnection



To reduce the load on the AHB bus matrix, some of the low-performance peripherals are connected through the synchronous AHB-to-AHB bridge with an address decoder. The AHB bus matrix is constructed of combinatorial logic, except for the AHB-to-AHB bridge, which inserts a one-cycle delay in each direction.

The following figure shows the block diagram of all the APB peripherals connected to AHB bus matrix using the AHB-to-APB bridge. The MSS APB peripherals are connected through the AHB to APB bus.

Figure 103 • Block Diagram of APB Destinations Connected to AHB Bus Matrix



7.1.2 Timing Diagrams

The following figures are the functional timing diagrams for AHBL read/write transactions through the AHB bus matrix and AHB-to-AHB bridge. Signals to/from a master are denoted by X in the signal name, and signals to/from a slave are denoted with Y in the signal name. For example, if Cortex-M3 processor master initiates the transactions of read/write to the eSRAM slave then the signals with X in the signal name indicates the signals of the Cortex-M3 processor and signals with Y indicate slave eSRAM signals.

Figure 104 • AHB-Lite Write Transactions

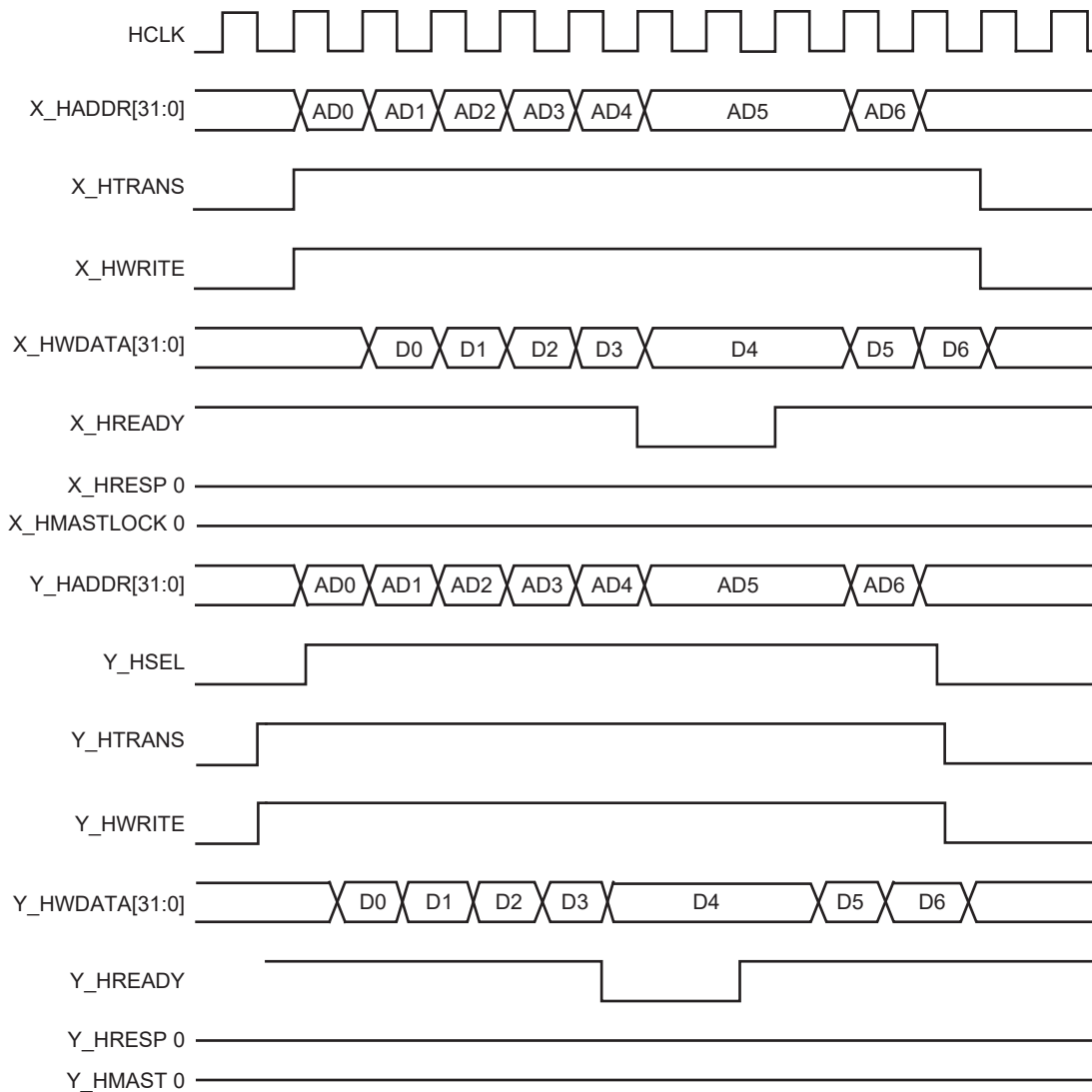


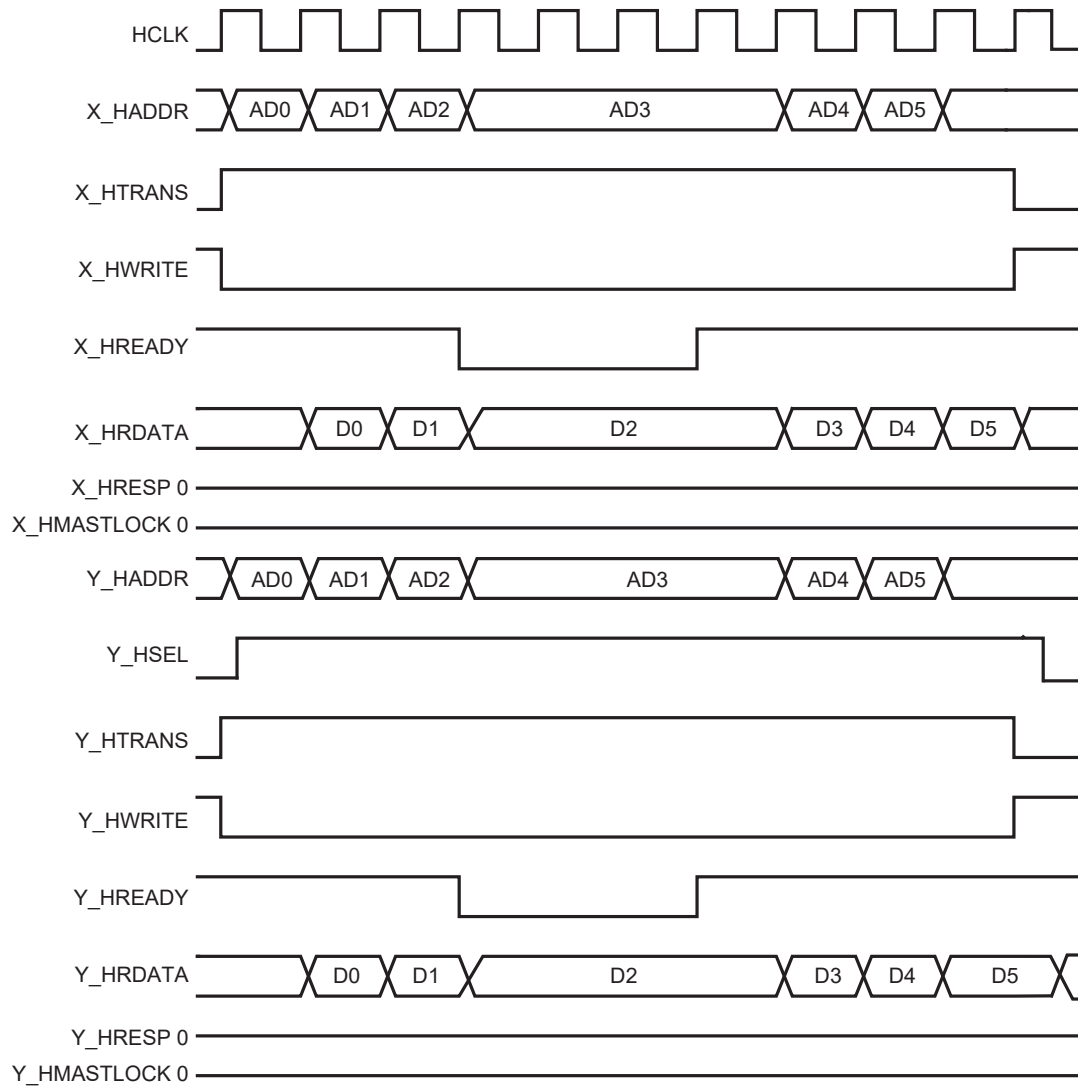
Figure 105 • AHB-Lite Read Transactions

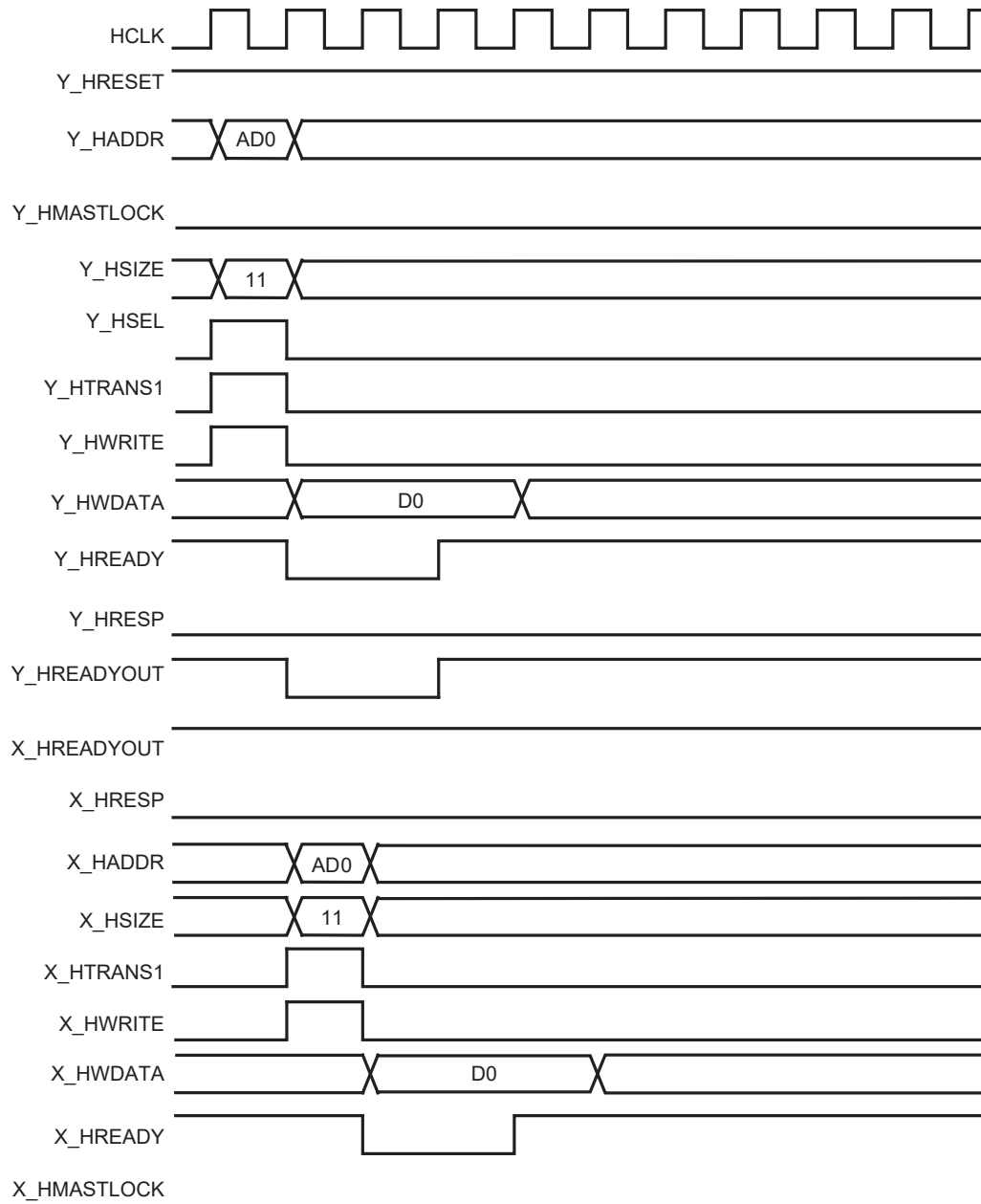
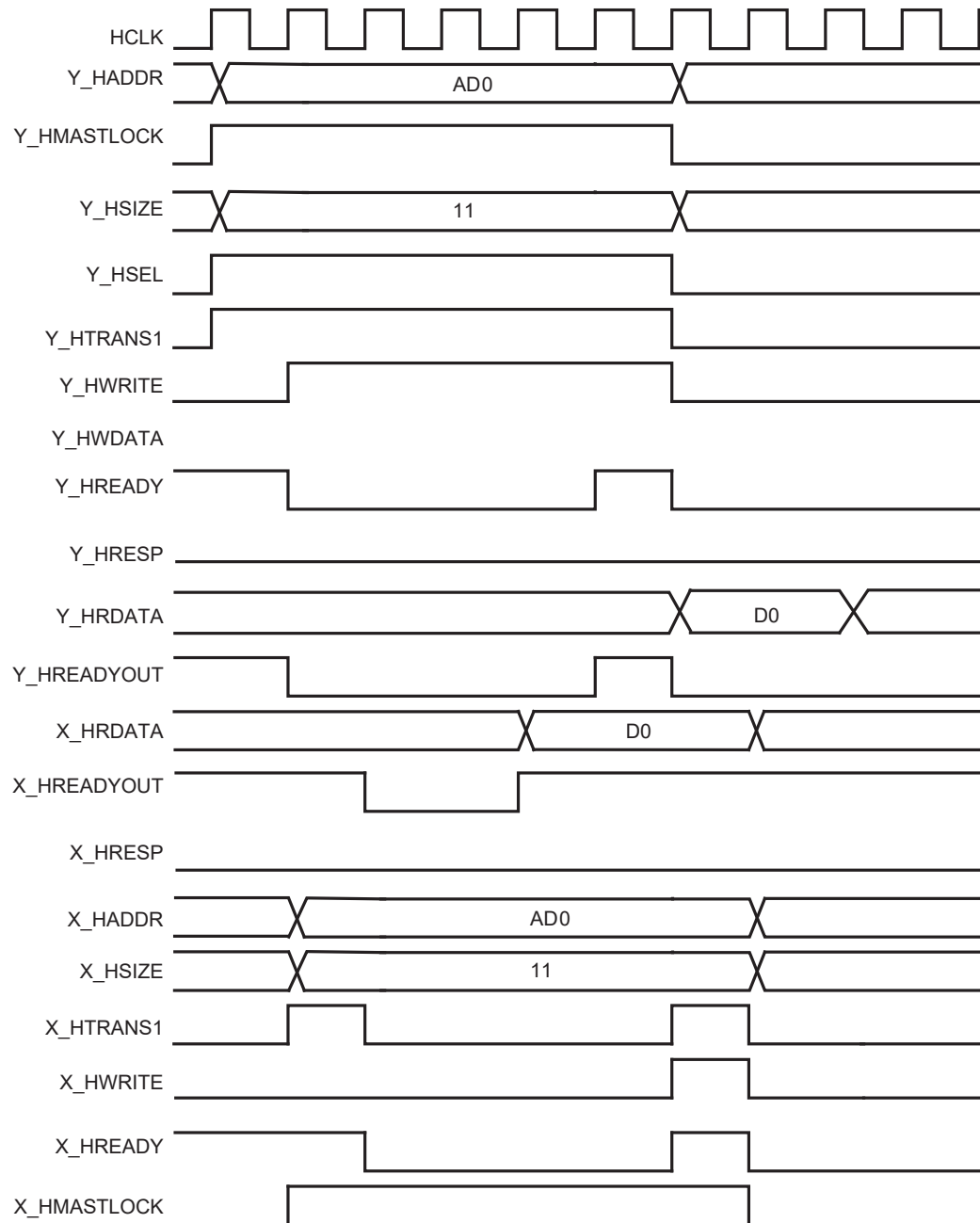
Figure 106 • AHB-to-AHB Write Transactions

Figure 107 • AHB-to-AHB Read Transactions

7.1.3 Details of Operation

7.1.3.1 Slave Arbitration

Each of the slave devices on the AHB bus matrix contains an arbiter. Arbitration is done at two levels. At the first level, the fixed higher priority masters are evaluated for any access request to the slave. At the second level, the remaining busses are evaluated in round robin fashion for any access request to the slave. The priority levels of the busses with fixed priority are listed in the following table.

Table 140 • Fixed Priority Masters

Masters		Priority	Arbitration
M3 DCode bus	MM0	1	Fixed
M3 ICode bus	MM1	2	Fixed
M3 system bus	MM2	3	Fixed
System controller	MM9	4	Fixed

The buses with round robin priority are listed in the following table.

Table 141 • WRR Masters

Masters		Priority	Arbitration
HPDMA	MM3	4	WRR
FIC_0	MM4	4	WRR
FIC_1	MM5	4	WRR
MAC	MM6	4	WRR
PDMA	MM7	4	WRR
USB	MM8	4	WRR

7.1.3.1.1 Arbitration Parameters

The following slave arbitration configuration parameters are user programmable registers in the SYSREG block.

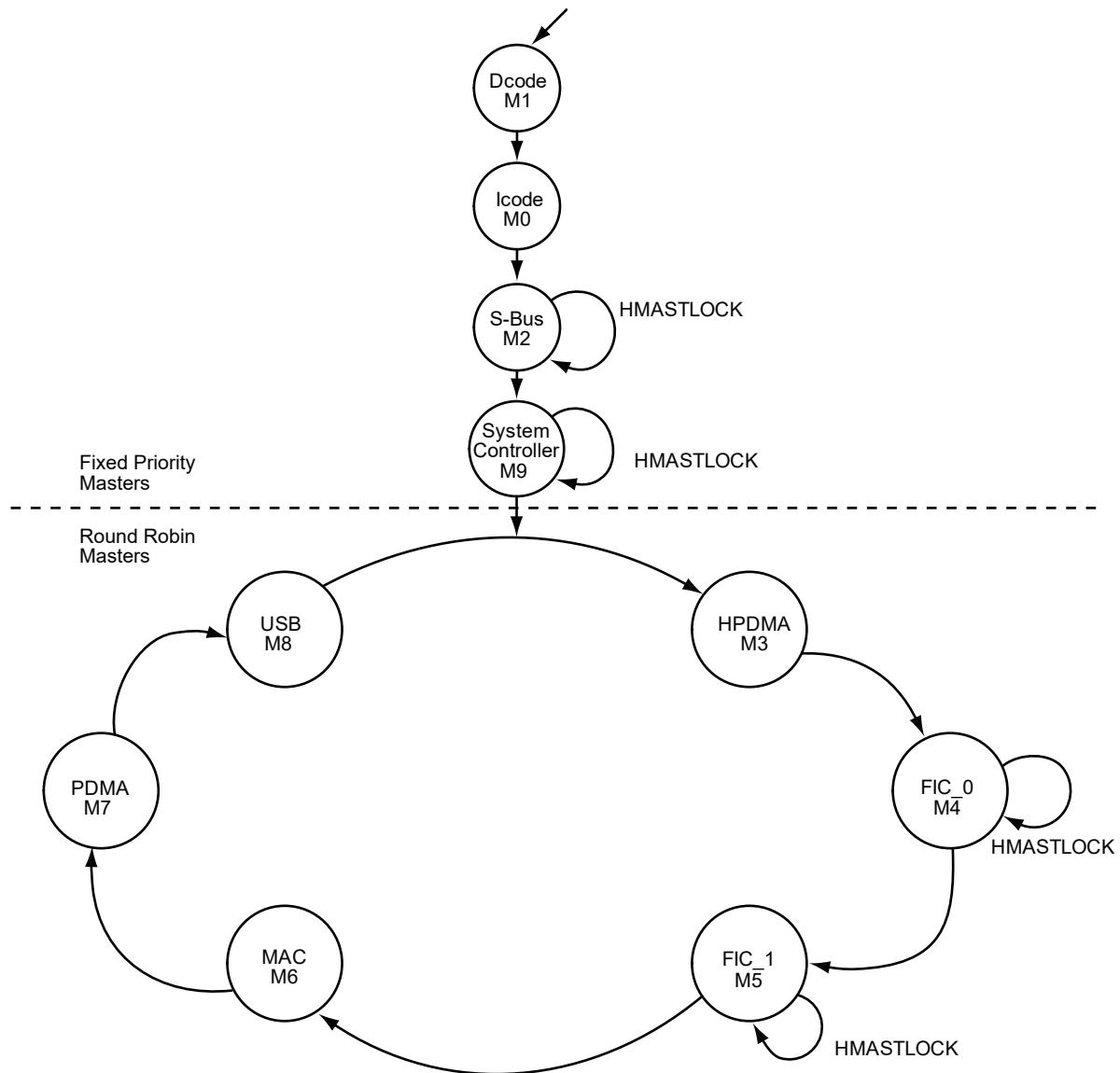
- **Programmable slave maximum latency:** Slave maximum latency, ESRAM_MAX_LAT, decides the peak wait time for a fixed priority master arbitrating for eSRAM access while the WRR master is accessing the slave. After the defined latency period, the WRR master will have to re-arbitrate for slave access. Slave maximum latency can be configurable from one to eight clock cycles (eight by default). ESRAM_MAX_LAT is only supported for fixed priority masters addressing eSRAM slaves; it has no effect on WRR masters. The system designer can use this feature to ensure the processor latency for accesses to eSRAM is limited to a defined number of clock cycles. This is to facilitate limiting the ISR latency for real-time-critical functions.
- **Programmable weight:** MASTER_WEIGHT0_CR and MASTER_WEIGHT1_CR are 5-bit programmable registers located in the SYSREG block that define the number of consecutive transfers the weighted master can perform without being interrupted by a fixed priority master, or before moving onto the next master in the WRR cycle.

7.1.3.1.2 Pure Round Robin Arbitration

This is the default arbitration mode after reset. The programmable weight value of each master is set to 1, and ESRAM_MAX_LAT = 1.

The arbitration scheme for each slave port is identical in pure round robin arbitration, as shown in the following figure. The processor masters have priority over the non-processor masters. Each non-processor master accessing a slave has equal priority on a round robin basis. However, if a locked transaction occurs, the master issuing the lock maintains ownership of the slave until the locked transaction completes.

Figure 108 • Pure Round Robin and Fixed Priority Slave Arbitration Scheme



The following table gives an example of a pure round robin and fixed priority arbitration scenario for eSRAM1. This example illustrates default AHB bus matrix behavior.

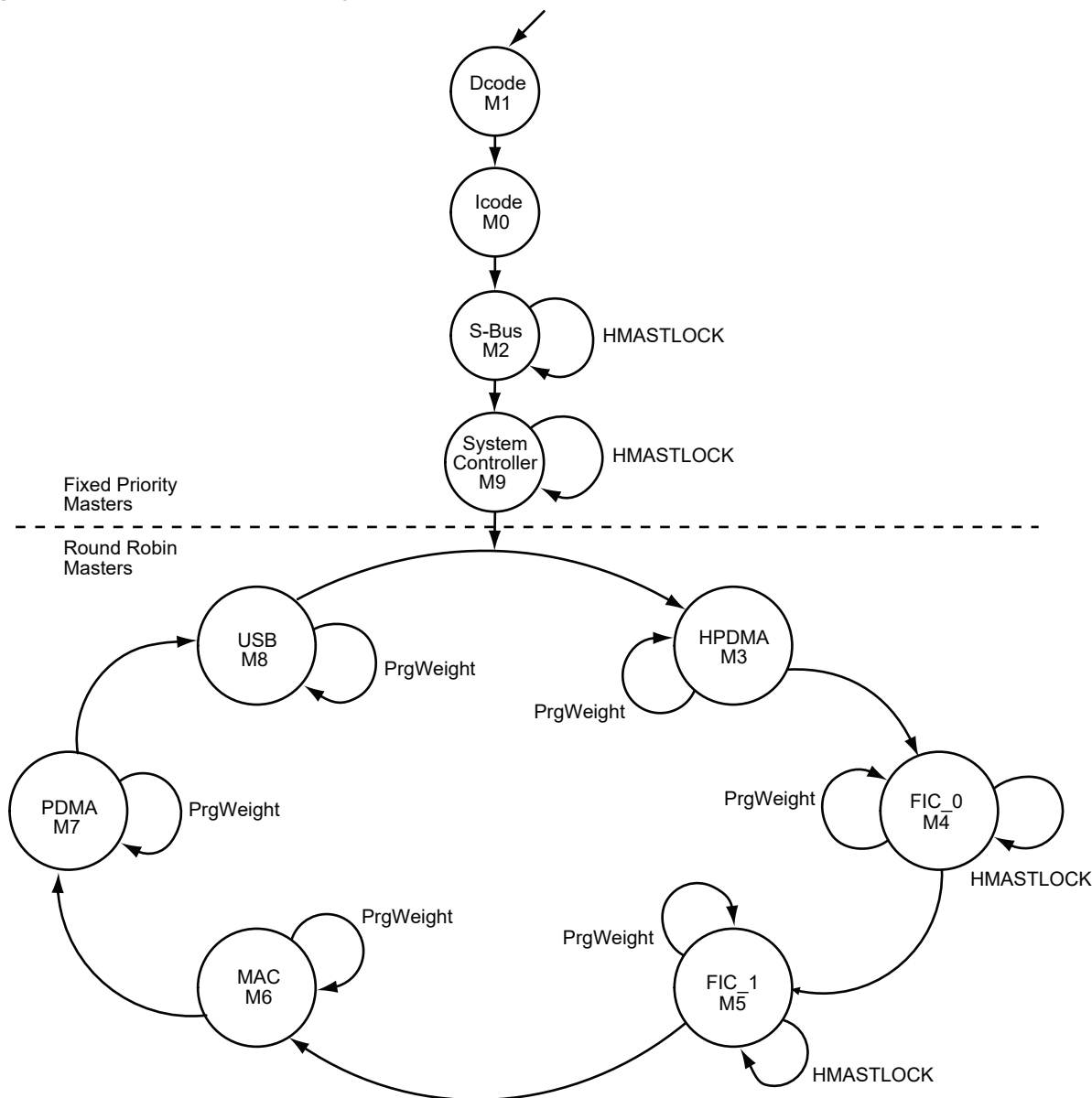
Table 142 • Pure Round Robin and Fixed Priority Arbitration Scenario for eSRAM1

Master	HCLK								
	1	2	3	4	5	6	7	8	9
M3-I: M0				eSRAM1					
M3-D: M1				eSRAM1					
M3-S: M2		eSRAM1			eSRAM1				
HPDMA: M3	eSRAM1								
FIC_0: M4	eSRAM1								
FIC_1: M5	eSRAM1								
GIGE: M6	eSRAM1								
PDMA: M7	eSRAM1								
eSRAM1: S1	HPDMA M3	M3-S M2	FIC_0 M4	M3-D M1	M3-I M0	M3-S M2	FIC_1 M5	TSE M6	PDMA M7

In the preceding table, WRR masters and fixed priority masters arbitrate for the S1 (eSRAM1) slave during HCLK cycle 1. The last row in the table, labeled eSRAM1: S1, shows which of the masters obtains access to the slave according to the arbitration in that clock cycle. In the first cycle, master M3 (HPDMA) is granted access, since it is the first master in the round robin scheme. In the second cycle, even though master M4 is scheduled to get access to the slave as per the round robin scheme, the M2 master (Cortex-M3 processor SBus) is granted access since it has a higher priority. In the third cycle, the master M4 (FIC_0) in the round robin scheme is granted access. In the fourth cycle, both M0 (Cortex-M3 processor ICode bus) and M1 (Cortex-M3 processor DCode bus) are trying for access. Since M1 has the highest priority among the fixed priority masters, it is granted access, followed by the M0 master. WRR masters are delayed while the fixed priority masters get access to the slave. The remaining cycles are consumed by the WRR masters in order.

7.1.3.1.3 WRR Arbitration

In this mode, the slave arbitration parameters, programmable weight (SW_WEIGHT_<master>) and eSRAM slave maximum latency (SW_MAX_LAT_ESRAM<0/1> of ESRAM_MAX_LAT) can be configured to operate as WRR arbitration. The slave arbiter operates on a round robin basis, with each master having a maximum of N consecutive access opportunities to the slave in each round of arbitration. The value of N is determined by the programmed weight for the master and eSRAM slave maximum latency. Programmable weight values can be changed dynamically. The following figure depicts the WRR slave arbitration scheme. At each stage, the arbiter checks whether that master is requesting access. If so, the master performs N transfers equal to its programmed weight and then has to re-arbitrate for the bus. For a WRR master, the WRR priority in the round robin sequence changes after the programmed number of transfers. Due to its highest priority, the Cortex-M3 processor DCode bus master is allowed to perform transfers as long as there are transfers to complete. However, if a locked transaction occurs, the master issuing the lock (HMASTLOCK = 1) maintains ownership of the slave until the locked transaction completes.

Figure 109 • WRR and Fixed Priority Slave Arbitration Scheme

WRR with fixed priority arbitration allows more efficient usage of slave bandwidth in cases where the slaves have a penalty when transitioning from one master to another. For example, in situations where both the Ethernet MAC and Cortex-M3 processor ICode/DCode interfaces are performing write and read AHB bursts to eSRAM, this scheme groups together a maximum of N Ethernet MAC accesses followed by a maximum of M Cortex-M3 processor accesses (even if AHB bursts of greater than N or M transfers are in progress from the master's point of view). Due to the fact that the eSRAM AHB controller inserts an idle cycle every time there is a write followed by a read, enabling WRR can increase the effective eSRAM bandwidth during this time from 66% to 94% of the theoretical maximum. If a sequence of locked transfers is in progress, the locked master remains selected by the slave arbiter until the lock sequence is finished, regardless of the number of transfers. For the case described, the values of N and M are SW_WEIGHT_MAC and SW_WEIGHT_HPDM in the MASTER_WEIGHT0_CR control register.

Arbitration for Non-eSRAM Slaves

In non-eSRAM slaves, any WRR master getting access to the slave can perform uninterrupted transactions equal to its programmed weight before re-arbitrating for the slave. Thus, for example, if FIC_1 is programmed with a weight of 8, it can do 8 continuous transactions with the slave even if the

high priority master is requesting access to the slave. Only after completing 8 transfers, the high priority master will gain access to the slave.

The following table gives an arbitration scenario for a non-eSRAM slave. In this scenario, master M5 (FIC_1) starts a burst of twelve transfers (reads typically for accesses to eNVM) to slave S2 (eNVM_0) in the first clock cycle. In the second clock, fixed priority master M1 (DCode bus) tries to access the same slave. Since the programmed weight of M5 master is 8, the M1 master does not gain access to the slave until M5 completes eight transfers. As seen in the table, the M1 master gains access to the slave only after the M5 master completes eight transfers, which is in the 9th clock cycle. The M5 master has to re-arbitrate for the slave to complete the remaining transfers. So the maximum latency seen by the Cortex-M3 processor bus M1 is equal to the programmed weight of 8.

Table 143 • WRR and Fixed Priority Arbitration Scenario for eNVM_0

Master	HCLK															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
M3-D: M1		S2-B4														
FIC_1: M5	S2-B12															
eNVM_0:S2	M5-B1	M5-B2	M5-B3	M5-B4	M5-B5	M5-B6	M5-B7	M5-B8	M1-B1	M1-B2	M1-B3	M1-B4	M5-B9	M5-B10	M5-B11	M5-B12

7.1.3.1.4 Arbitration for eSRAM Slaves

For eSRAM slaves, the maximum latency seen by Cortex-M3 processor bus masters can affect overall system performance. To manage this latency, a programmable maximum latency parameter `SW_MAX_LAT_ESRAM<0/1>` is available to optimize arbitration for the eSRAM slaves from a fixed priority master. The parameter `SW_MAX_LAT_ESRAM<0/1>` sets a ceiling as to the number of cycles the Cortex-M3 processor bus master, or any fixed priority master, has to wait before accessing an eSRAM slave that is currently being accessed by a WRR master. When a WRR master has a programmable weight greater than the `SW_MAX_LAT_ESRAM<0/1>` value, the WRR master will have to re-arbitrate for the slave after `SW_MAX_LAT_ESRAM<0/1>` cycles. The following equation gives the maximum latency seen by a processor master while accessing an eSRAM slave:

Maximum latency seen by the Cortex-M3 processor master = min {programmable weight (WRR master), `SW_MAX_LAT_ESRAM<0/1>`}

For example, if `SW_WEIGHT_HPDM` is set to 18 and `SW_MAX_LAT_ESRAM0` is set to 4, then the maximum latency is min {18, 4} = 4. Similarly, if `SW_WEIGHT_PDMA` is set to 2 and `SW_MAX_LAT_ESRAM1` is set to 6, then the maximum latency is min {2, 6} = 2. The following table depicts a typical scenario.

Table 144 • WRR Arbitration Scenario for eSRAM_0 slave

Master	HCLK											
	1	2	3	4	5	6	7	8	9	10	11	12
M3-D: M1			S0-B4									
PDMA: M7	S0-B8											
eSRAM_0: S0	M7-B1	M7-B2	M7-B3	M7-B4	M1-B1	M1-B2	M1-B3	M1-B4	M7-B5	M7-B6	M7-B7	M7-B8

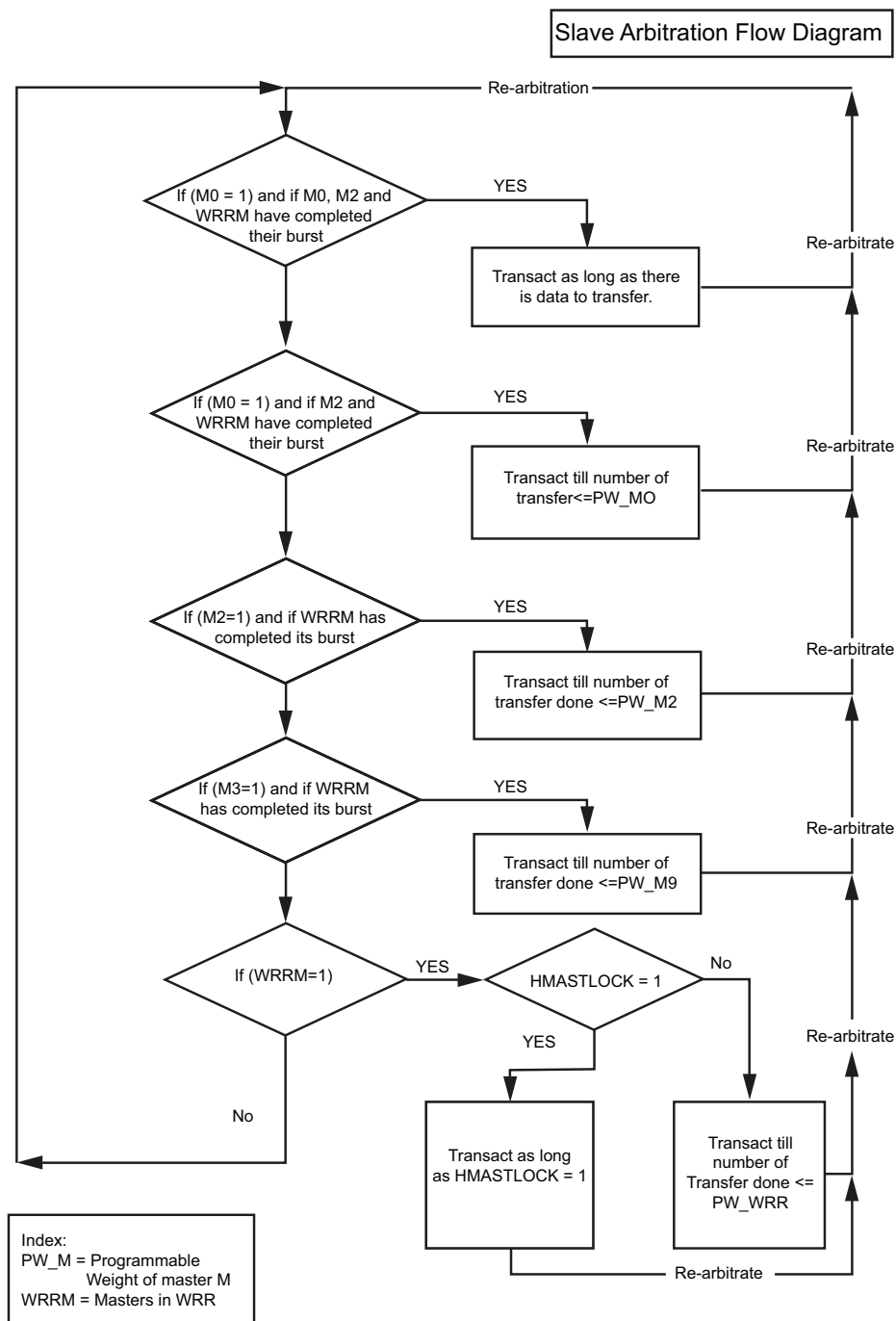
In this scenario, the slave maximum latency is set to 4 and the master programmable weight is set to 8, so the maximum latency seen by the processor bus master is min {4, 8} = 4. When the WRR master starts transactions with the eSRAM slave, it can perform a number of transactions equal to the programmed maximum latency or the programmed weight, whichever is less, before re-arbitrating for the slave.

7.1.3.1.5 Slave Arbitration Flow Diagram

The following figure shows the slave arbitration flow diagram depicting the grant of access to master requesting for slave access. At each stage the arbiter checks whether that master is requesting for an access. If yes, then the master can do number of transfers equal to its programmed weight and then has to re-arbitrate for the bus. In case of WRRM master, after the programmed number of transfer WRRM priority changes for that master in round robin sequence.

The D-Code master can do transfers as long as there are transfers to complete. In case of lock transfers, master issuing lock (HMASTLOCK = 1) maintains the ownership of the slave as long as lock signal is asserted.

Figure 110 • Slave Arbitration Flow Diagram



7.1.3.2 HBURST Support for eNVM

The AHB bus matrix only supports AHB bursts from the Cortex-M3 processor bus or from the cache to the eNVM. To support burst reads from the eNVM, you should program SW_WEIGHT_IC to be the maximum burst expected. SW_WEIGHT_IC = 32 allows a cache line to be filled from eNVM without interruption.

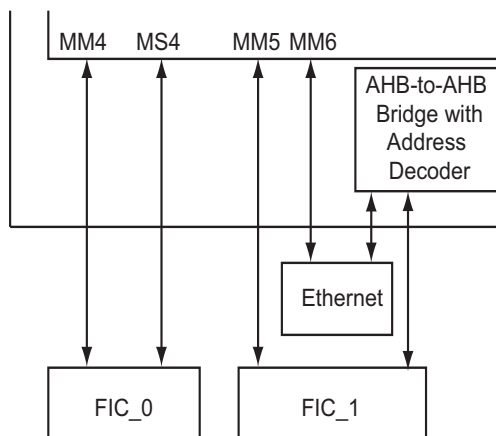
7.1.3.3 AHB Bus Matrix to Fabric Interface Controller

The AHB bus matrix provides two 2-bit side band signals—driven out of the AHB bus matrix to the fabric interrupt controller to indicate which master is asserting FIC_X (X indicates FIC 0 or 1 master). The following figure shows the two FIC blocks connected to AHB bus matrix.

The FIC block provides two separate interfaces between the MSS and the FPGA fabric: the hard master (HM) and fabric master (FM). These interfaces may be configured to operate as AHB32 or APB32. Configure FIC_0 and FIC_1 interfaces in bypass mode to perform weighted round robin arbitration. For more information, refer to the [AC388: SmartFusion2 SoC FPGA - Dynamic Configuration of AHB Bus Matrix Application Note](#).

FIC_0_MASTER_ID and FIC_1_MASTER_ID are two signals from FIC_0 and FIC_1 to FPGA fabric that indicate the current master group accessing fabric slaves. These two signals are exposed when each FIC block interface towards the fabric is configured with the Libero SoC MSS configurator.

Figure 111 • AHB Bus Matrix to Fabric Interface Controller



The following table provides the decoding of master access done by the AHB bus matrix to the fabric slave.

Table 145 • Decoding of Master Access to the Fabric Slaves

FIC_X_MASTER_ID	Accessing Master
00	IC-bus, D-bus, and S-bus master
01	FIC_0, FIC_1 master
10	HPDMA, Ethernet master, PDMA, USB
11	System controller

7.1.4 System Memory Map

The AHB bus matrix is responsible for implementing the address decoding of all masters to all slaves, so it defines the system memory map. The following figure depicts the default system memory map for SmartFusion2 devices.

Figure 112 • Default System Memory Map

		Memory Map of System Controller, FPGA Fabric Master, Ethernet MAC, Peripheral DMA	
Memory Map of Cortex-M3 Processor		FPGA Fabric FIC Region5	
	FPGA Fabric FIC Region5		FPGA Fabric FIC Region5
	DDR_0 Space 3		DDR_0 Space 3
	DDR_0 Space 2		DDR_0 Space 2
	DDR_0 Space 1		DDR_0 Space 1
	DDR_0 Space 0		DDR_0 Space 0
	FPGA Fabric FIC Region4		FPGA Fabric FIC Region4
	FPGA Fabric FIC Region3		FPGA Fabric FIC Region3
	FPGA Fabric FIC Region2		FPGA Fabric FIC Region2
	AHB-to-eNVM_1 Registers		AHB-to-eNVM_1 Registers
	AHB-to-eNVM_0 Registers		AHB-to-eNVM_0 Registers
	eNVM_1		eNVM_1
	eNVM_0		eNVM_0
	FPGA Fabric FIC Region1		FPGA Fabric FIC Region1
Peripheral bit-band alias region of Cortex-M3 processor	Peripherals (BB View)		
	Cache Back door		
	USB		USB
	Ethernet MAC Control		Ethernet MAC Control
	SYSREG		SYSREG
	Config DDR_1, PCIe_0, PCIe_1, etc.		Config DDR_1, PCIe_0, PCIe_1, etc.
	Config DDR_0		Config DDR_0
	RTC		RTC
	COMBLK		COMBLK
	CAN		CAN
	High Performance DMA		High Performance DMA
	MSS GPIO		MSS GPIO
	I2C_1		I2C_1
	SPI_1		SPI_1
	UART_1		UART_1
	Fabric Interface Interrupt Controller		Fabric Interface Interrupt Controller
	Watchdog		Watchdog
	Timer		Timer
	Peripheral DMA Control		Peripheral DMA Control
	I2C_0		I2C_0
	SPI_0		SPI_0
	UART_0		UART_0
	FPGA Fabric FIC Region0		FPGA Fabric FIC Region0
SRAM bit-band alias region of Cortex-M3 processor	eSRAM_0/eSRAM_1 (BB View)		
	ECC eSRAM_1		ECC eSRAM_1
	ECC eSRAM_0		ECC eSRAM_0
Cortex-M3 processor system region	eSRAM_1		eSRAM_1
	eSRAM_0		eSRAM_0
Cortex-M3 processor code region	eNVM (Cortex-M3 processor) virtual view		eNVM (Fabric) virtual view

0xF0000000 - 0xFFFFFFFF
 0xE0000000 - 0xFFFFFFFF
 0xD0000000 - 0xFFFFFFFF
 0xC0000000 - 0xFFFFFFFF
 0xB0000000 - 0xFFFFFFFF
 0xA0000000 - 0xFFFFFFFF
 0x90000000 - 0xFFFFFFFF
 0x80000000 - 0xFFFFFFFF
 0x70000000 - 0xFFFFFFFF
 0x60100000 - 0xFFFFFFFF
 0x600C0000 - 0x600FFFFF
 0x60080000 - 0x600BFFFF
 0x60040000 - 0x6007FFFF
 0x60000000 - 0x6003FFFF
 0x50000000 - 0x5FFFFFFF
 0x44000000 - 0x4FFFFFFF
 0x42000000 - 0x43FFFFFF
 0x40410000 - 0x41FFFFFF
 0x40400000 - 0x4040FFFF
 0x40044000 - 0x403FFFFF
 0x40043000 - 0x40043FFF
 0x40042000 - 0x40042FFF
 0x40041000 - 0x40041FFF
 0x40039000 - 0x40040FFF
 0x40038000 - 0x40038FFF
 0x40030000 - 0x40037FFF
 0x40020400 - 0x4002FFFF
 0x40020000 - 0x400203FF
 0x40018000 - 0x4001FFFF
 0x40017000 - 0x40017FFF
 0x40016000 - 0x40016FFF
 0x40015000 - 0x40015FFF
 0x40014000 - 0x40014FFF
 0x40013000 - 0x40013FFF
 0x40012000 - 0x40012FFF
 0x40011000 - 0x40011FFF
 0x40010000 - 0x40010FFF
 0x40007000 - 0x4000FFFF
 0x40006000 - 0x40006FFF
 0x40005000 - 0x40005FFF
 0x40004000 - 0x40004FFF
 0x40003000 - 0x40003FFF
 0x40002000 - 0x40002FFF
 0x40001000 - 0x40001FFF
 0x40000000 - 0x40000FFF
 0x30000000 - 0x3FFFFFFF
 0x24000000 - 0x2FFFFFFF
 0x22000000 - 0x23FFFFFFF
 0x20014000 - 0x21FFFFFFF
 0x20012000 - 0x20013FFF
 0x20010000 - 0x20011FFF
 0x20008000 - 0x2000FFFF
 0x20000000 - 0x20007FFF
 0x00080000 - 0x1FFFFFFF

(63 K space
allocation for
devices outside
MSS)

} Visible only to
FPGA fabric
master

7.1.4.1 eSRAM Remap

The AHB bus matrix supports remapping the eSRAM address space into code space that Cortex-M3 processor can use. Both eSRAM blocks can be remapped to appear at the bottom of Cortex-M3 processor code space as shown in the preceding figure. The amount of space available to Cortex-M3 processor as code space depends on ECC as indicated below:

- When ECC is ON, the two eSRAM blocks (64 KB) can be remapped, but ECC sections of eSRAM (8 KB) cannot be used by the Cortex-M3 processor.
- When ECC is OFF, the two eSRAM blocks (64 KB) can be remapped, and the ECC sections of eSRAM (8 KB) can also be used by the Cortex-M3 processor. These 8 KB are available at a different address. The resultant memory map is illustrated in the following figure.

Figure 113 • Memory Map after eSRAM Remap (64 KB eSRAM)

Memory Map of Cortex-M3 Processor		Memory Map of System Controller, FPGA Fabric Master, Ethernet MAC, Peripheral DMA	
FPGA Fabric FIC Region5		FPGA Fabric FIC Region5	
DDR_0 Space 3		DDR_0 Space 3	
DDR_0 Space 2		DDR_0 Space 2	
DDR_0 Space 1		DDR_0 Space 1	
DDR_0 Space 0		DDR_0 Space 0	
FPGA Fabric FIC Region4		FPGA Fabric FIC Region4	
FPGA Fabric FIC Region3		FPGA Fabric FIC Region3	
FPGA Fabric FIC Region2		FPGA Fabric FIC Region2	
AHB-to-eNVM_1 Registers		AHB-to-eNVM_1 Registers	
AHB-to-eNVM_0 Registers		AHB-to-eNVM_0 Registers	
eNVM_1		eNVM_1	
eNVM_0		eNVM_0	
FPGA Fabric FIC Region1		FPGA Fabric FIC Region1	
Peripherals (BB View)			
Cache Back door			
USB		USB	
Ethernet MAC Control		Ethernet MAC Control	
SYSREG		SYSREG	
Config DDR_1, PCIe_0, PCIe_1, etc.		Config DDR_1, PCIe_0, PCIe_1, etc.	
Config DDR_0		Config DDR_0	
RTC		RTC	
COMBLK		COMBLK	
CAN		CAN	
High Performance DMA		High Performance DMA	
MSS GPIO		MSS GPIO	
I2C_1		I2C_1	
SPI_1		SPI_1	
UART_1		UART_1	
Fabric Interface Interrupt Controller		Fabric Interface Interrupt Controller	
Watchdog		Watchdog	
Timer		Timer	
Peripheral DMA Control		Peripheral DMA Control	
I2C_0		I2C_0	
SPI_0		SPI_0	
UART_0		UART_0	
FPGA Fabric FIC Region0		FPGA Fabric FIC Region0	
eSRAM_0/eSRAM_1(BB View)			
ECC eSRAM_1		ECC eSRAM_1	
ECC eSRAM_0		ECC eSRAM_0	
eSRAM_1		eSRAM_1	
eSRAM_0		eSRAM_0	
0x10000000 - 0x1FFFFFFF		DDR_0 Space 0(Mirrored)	
Cortex-M3 processor code region			
0x00180000 - 0x0FFFFFFF		eNVM (Remap View)	
0x00100000 - 0x00170000			
0x00100000 - 0x000FFFFF		eSRAM0 and eSRAM1 (Mirrored)	
0x00000000 - 0x0000FFFF			
			0x0007FFFF
			0x00000000

Peripheral bit-band alias region of Cortex-M3 processor

SRAM bit-band alias region of Cortex-M3 processor

Cortex-M3 Processor System region

0x0007FFFF

0x00000000

(63 K space allocation for devices outside MSS)

Visible only to FPGA fabric master

In default mode, the Cortex-M3 processor firmware boots from eNVM. However, as shown in the preceding figure, it is also possible to get the firmware to boot from eSRAM by re-mapping eSRAM to location zero. Code shadowing is supported to facilitate this.

A master in the FPGA fabric must extend the assertion of reset to the Cortex-M3 processor until the system reset to the remainder of the MSS is negated. This master must then copy the appropriate code from eNVM to eSRAM and release the reset of the Cortex-M3 processor.

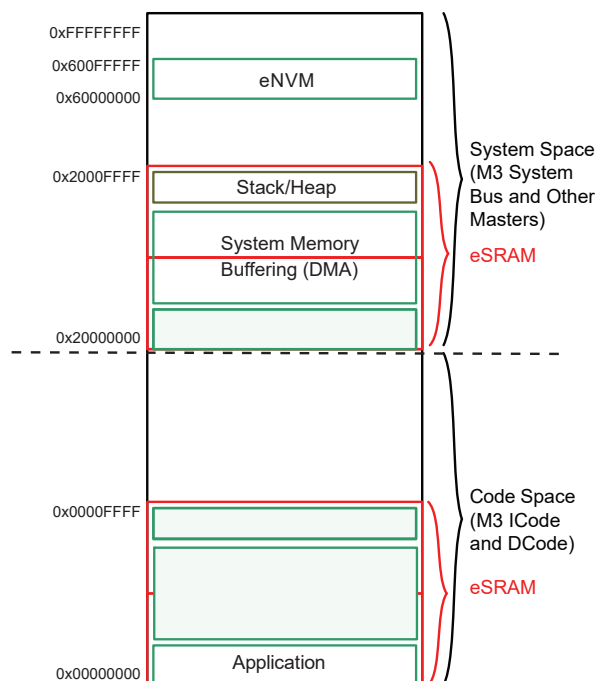
7.1.4.1.1 Executing from eSRAM

The eSRAM remap is actually performed by aliasing the eSRAM blocks so they appear in the Cortex-M3 processor code space but are still accessible in the Cortex-M3 processor system space. Therefore, the system designer must allocate the eSRAM among clients in such a way that a portion of eSRAM allocated in one space (code space, for example) is left untouched in the other space (system space, for example).

The Cortex-M3 processor executes the application (including ISRs) from the code space, allowing optimal performance. However, the corresponding region in system space is grayed out. Conversely, the stack (and heap, if present) as well as buffering for non-M3 masters (such as peripheral DMA or Ethernet DMA) is allocated out of system space and so must be left grayed out in the code space.

This implementation is shown in the following figure.

Figure 114 • Use Case for eSRAM Execution



This scheme allows flexibility to the system designer in choosing how much eSRAM is to be dedicated to each class of storage. For example, if the application, stack, and heap are small, this allows a large chunk of contiguous RAM to be allocated to buffering.

7.1.4.1.2 Using Harvard Architecture

When a system designer is more interested in optimal performance than flexibility, eSRAM_0 can be dedicated to the application (and ISRs), and eSRAM_1 can be dedicated to stack, heap, and buffering. This implies that the Cortex-M3 processor operates in a fully Harvard fashion, because eSRAM_0 can be accessed only by the combined code bus, and eSRAM_1 can be accessed by the system bus of M3 as well as the other (non-M3) masters.

7.1.4.1.3 Ensuring Deterministic Latency

If the system designer wishes to have deterministic latencies of ISR execution, the ISRs need to be located in eSRAM. The eSRAM must be un-contended in order to guarantee true determinism. A maximum latency value can be programmed to eSRAM slaves, so processor masters do not wait long for access to eSRAM slaves. The maximum latency is set by writing into SW_MAX_LAT_ESRAM0[2:0] or SW_MAX_LAT_ESRAM1 of ESRAM_MAX_LAT. You can execute code from the external memory

(SRAM or DDR). DDR is accessed by Cortex-M3 processor masters directly through the cache controller block and not through the AHB bus matrix block.

7.1.4.2 eNVM Remap

7.1.4.2.1 eNVM Remap for Cortex-M3

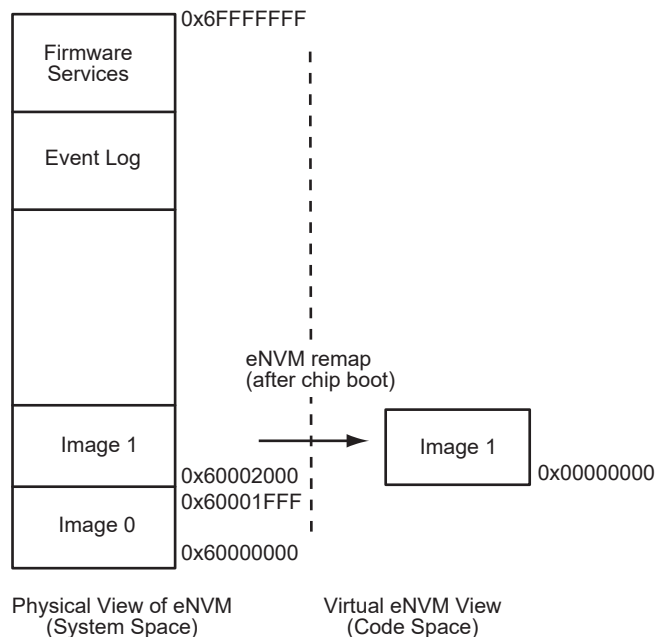
An eNVM can have multiple firmware images located at any of the possible locations of the eNVM array. But any image can be accessed from the zero or base address location in the virtual view by remapping the eNVM. The AHB bus matrix, under the control of the SYSREG block, handles the eNVM remap, whereby a virtual eNVM view is presented to the AHB bus for accesses in the range from 0x00000000 to 0x1FFFFFFF.

Of all masters in a SmartFusion2 device, only the Cortex-M3 processor ICode and DCode AHB busses access the eNVM in this range, which corresponds to its code space.

After reset, the AHB bus matrix must address the data at its physical address in the system memory map because it is not visible in the virtual map at this point. At the end of the boot process, the Cortex-M3 processor firmware writes into two SYSREG registers ENV_M_CR and ENV_REPMAP_BSASE_CR, which control eNVM remapping for the Cortex-M3 processor. This causes a specific segment of the eNVM array, of a specified size, to be remapped in the virtual view starting at address 0x00000000. This implies that if multiple firmware images are present, each may be built with the assumption that they are located at 0x00000000.

For example, consider that there are two firmware images: image0 and image1. After remap of image1, the virtual view of this image will be from 0x00000000 even though the physical address of the image starts at 0x60002000. The following figure shows this example representation of an eNVM remap after chip boot.

Figure 115 • Virtual eNVM View (After Chip Boot)



At the bottom of the remapped firmware image is the user boot code. This can call out to firmware services located at the top of eNVM. However, these services must be addressed by the firmware using the physical address of the firmware services.

7.1.4.2.2 eNVM Remap for Soft Processor

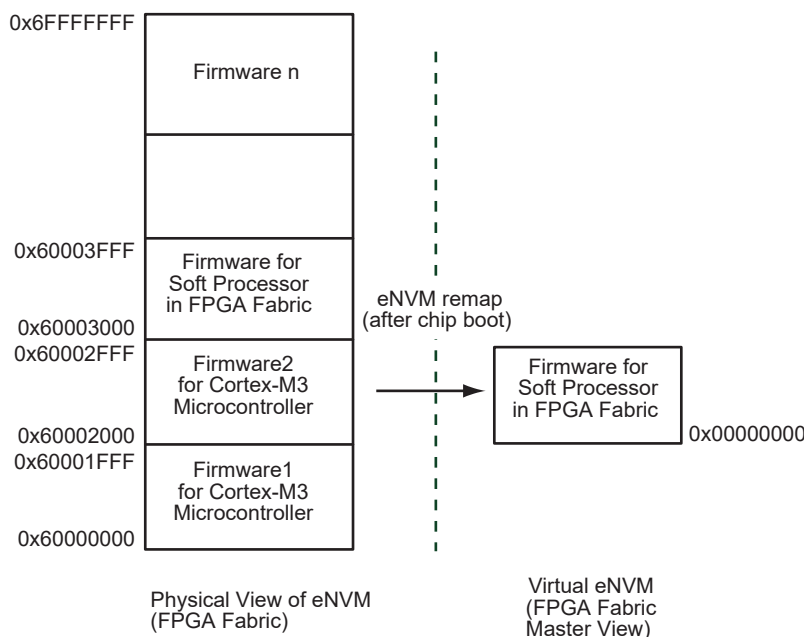
Any soft processor implemented within the FPGA fabric usually tries to fetch instructions from location 0x00000000. However, this refers to different code than the Cortex-M3 processor boot code, which resides at location 0x00000000 as far as the Cortex-M3 processor is concerned. The AHB bus matrix

supports remapping of an eNVM segment to location 0x00000000 in the memory map seen by masters in the FPGA fabric with the ENVM_REMAP_FAB_CR control register.

The ENVM_REMAP_FAB_CR control register configures where ENVM is mapped in fabric space. There is no eSRAM remap for fabric masters. Therefore, a fabric master must always access eSRAM from its location described in [Figure 112](#), page 225.

The following figure gives an example representation of the eNVM remap of a soft processor implemented in FPGA fabric and remapped to a virtual address 0x00000000, even though its physical address starts at 0x60003000.

Figure 116 • Virtual eNVM View for Soft Processor



7.1.4.3 DDR Memory Map

Although up to 4 Gbytes of DDR is supported by the system, only 1 GB of this is accessible at one time from the Cortex-M3 processor or MSS masters via the AHB bus matrix. The HPDMA and DDR_FIC can access all 4 Gbytes at default settings. In order to make a particular region of DDR visible to the Cortex-M3 processor firmware or another non-HPDMA MSS master, it is necessary to configure the appropriate DDR mapping registers in the MSS system registers.

7.1.4.4 DDR Remap

In default mode, the Cortex-M3 processor firmware boots from eNVM. However, as shown in the following figure, it is also possible to get the firmware to boot from DDR by re-mapping DDR to location zero. Code shadowing is supported to facilitate this. User boot firmware, located in eNVM, must copy an executable image from external flash memory (serial or parallel) to external DDR memory, then jump to the application entry point in external DDR memory.

In DDR remap mode, the total available cacheable region (512 Mbytes) can be configured to 128 Mbytes, 256 Mbytes, or 512 Mbytes. In the case of a 128 Mbyte cacheable size, the entire 512 Mbytes is divided into four cacheable regions of 128 Mbytes each, and one of the four regions will be selected as per configuration. Similarly for 256 Mbytes, one of the two cacheable regions (512 Mbytes cacheable region split to two 256 Mbyte regions) will be selected as per configuration. These selections can be configured using the DDRB_NB_ADDR_CR and DDRB_NB_SIZE_CR registers.

The cache controller generates the appropriate DDR address as per remap before putting the access request to the MSS DDR bridge. A soft SDRAM memory controller implemented in the fabric can be remapped to address 0 just like the MDDR so that external code located in SDRAM is cacheable.

Figure 117 • DDR Memory Remap

		Memory Map of Cortex-M3 Processor	Memory Map of System Controller, FPGA Fabric Master, Ethernet MAC, Peripheral DMA	
		FPGA Fabric FIC Region5	FPGA Fabric FIC Region5	0xF0000000 - 0xFFFFFFFF
				0xE0000000 - 0xEFFFFFFF
		MDDR Space 3	DDR_0 Space 3	0xD0000000 - 0xDFFFFFFF
		MDDR Space 2	DDR_0 Space 2	0xC0000000 - 0xCFFFFFFF
		MDDR Space 1	DDR_0 Space 1	0xB0000000 - 0xBFFFFFFF
		MDDR Space 0	DDR_0 Space 0	0xA0000000 - 0xAFFFFFFF
		FPGA Fabric FIC Region4	FPGA Fabric FIC Region4	0x90000000 - 0x9FFFFFFF
		FPGA Fabric FIC Region3	FPGA Fabric FIC Region3	0x80000000 - 0x8FFFFFFF
		FPGA Fabric FIC Region2	FPGA Fabric FIC Region2	0x70000000 - 0x7FFFFFFF
				0x60100000 - 0x6FFFFFFF
		AHB-to-eNVM_1 Registers	AHB-to-eNVM_1 Registers	0x600C0000 - 0x600FFFFF
		AHB-to-eNVM_0 Registers	AHB-to-eNVM_0 Registers	0x60080000 - 0x600BFFFF
		eNVM_1	eNVM_1	0x60040000 - 0x6007FFFF
		eNVM_0	eNVM_0	0x60000000 - 0x6003FFFF
		FPGA Fabric FIC Region1	FPGA Fabric FIC Region1	0x50000000 - 0x5FFFFFFF
				0x44000000 - 0x4FFFFFFF
		Peripherals (BB View)		0x42000000 - 0x43FFFFFF
				0x40410000 - 0x41FFFFFF
		Cache Back door		0x40400000 - 0x4040FFFF
				0x40044000 - 0x403FFFFF
		USB	USB	0x40043000 - 0x40043FFF
				0x40042000 - 0x40042FFF
		Ethernet MAC Control	Ethernet MAC Control	0x40041000 - 0x40041FFF
				0x40039000 - 0x40040FFF
		SYSREG	SYSREG	0x40038000 - 0x40038FFF
				0x40030000 - 0x40037FFF
		Config DDR_1, PCIe_0, PCIe_1, etc.	Config DDR_1, PCIe_0, PCIe_1, etc.	0x40020400 - 0x4002FFFF
		Config DDR_0	Config DDR_0	0x40020000 - 0x400203FF
				0x40018000 - 0x4001FFFF
		RTC	RTC	0x40017000 - 0x40017FFF
		COMBLK	COMBLK	0x40016000 - 0x40016FFF
		CAN	CAN	0x40015000 - 0x40015FFF
		High Performance DMA	High Performance DMA	0x40014000 - 0x40014FFF
		MSS GPIO	MSS GPIO	0x40013000 - 0x40013FFF
		I2C_1	I2C_1	0x40012000 - 0x40012FFF
		SPI_1	SPI_1	0x40011000 - 0x40011FFF
		UART_1	UART_1	0x40010000 - 0x40010FFF
				0x40007000 - 0x4000FFFF
		Fabric Interface Interrupt Controller	Fabric Interface Interrupt Controller	0x40006000 - 0x40006FFF
		Watchdog	Watchdog	0x40005000 - 0x40005FFF
		Timer	Timer	0x40004000 - 0x40004FFF
		Peripheral DMA Control	Peripheral DMA Control	0x40003000 - 0x40003FFF
		I2C_0	I2C_0	0x40002000 - 0x40002FFF
		SPI_0	SPI_0	0x40001000 - 0x40001FFF
		UART_0	UART_0	0x40000000 - 0x40000FFF
		FPGA Fabric FIC Region0	FPGA Fabric FIC Region0	0x30000000 - 0x3FFFFFFF
				0x24000000 - 0x2FFFFFFF
		eSRAM_0/eSRAM_1(BB View)		0x22000000 - 0x23FFFFFF
				0x20014000 - 0x21FFFFFF
		ECC eSRAM_1	ECC eSRAM_1	0x20012000 - 0x20013FFF
		ECC eSRAM_0	ECC eSRAM_0	0x20010000 - 0x20011FFF
		eSRAM_1	eSRAM_1	0x20008000 - 0x2000FFFF
		eSRAM_0	eSRAM_0	0x20000000 - 0x20007FFF
				0x00080000 - 0x1FFFFFFF
		MDDR Space 1 (mirrored)		0x0007FFFF
		MDDR Space 0 (mirrored)	eNVM (Fabric) virtual view	
				0x00000000

Peripheral Bit-band alias region of Cortex-M3 processor

SRAM bit-band alias region of Cortex-M3 processor

Cortex-M3 processor system region

Cortex-M3 processor code region
0x10000000 - 0x1FFFFFFF

0x00000000 - 0x0000FFFF

(63 K space allocation for devices outside MSS)

Visible only to FPGA fabric master

7.1.4.5 Unimplemented Address Space

The AHB bus matrix performs address decoding based on the memory map defined in [Figure 112](#), page 225 and [Figure 113](#), page 226, to decide which slave, if any, is being addressed. Any access to memory space outside of these regions is considered unimplemented from the point of view of the AHB bus matrix. This access results in the assertion of a SW_ERRORSTATUS register bit of the MSS_EXTERNAL_SR control register, as well as the assertion of HRESP by the AHB bus matrix to the master. If any master attempts a write access to unimplemented address space, the AHB bus matrix completes the handshake to the master with an HRESP error indication. No write occurs to any slave.

If any master attempts a read access from unimplemented address space, the AHB bus matrix completes the handshake to the master with an HRESP error indication. Undefined data is returned. There may be further memory areas that are unimplemented, within individual slave memory regions. Depending on the slave, accesses may be aliased within these areas or not. Firmware should not perform writes to these locations because the aliasing may cause a write to another location within the slave. Data read from these intra-slave unimplemented regions may be undefined.

7.1.4.6 Burst Support

AHB-Lite HBURST is supported only for an ICode (IC) bus master accessing eNVM slaves. The AHB bus matrix handshakes correctly with masters performing AHB bursts to any slave. However, it does not pass the transactions through to the slaves as bursts. Instead, the AHB bus matrix converts the burst accesses into single-cycle accesses of the type NONSEQ. This simplifies the design of the slaves (which can exist in the FPGA fabric), since they do not need to support AHB bursts. It also allows the system designer to avoid long latencies incurred by bursts of indeterminate length (such as those from the FPGA fabric).

7.1.4.7 Locked Transactions

MSS supports locked accesses through its internal switch matrix to its slaves (eSRAM, DDR, FIC_0, and FIC_1). HMASTLOCK signal is not routed to the fabric to allow a matrix to implement a lock-based arbitration system.

7.1.4.8 Peripheral Bit-Banding

All of the peripherals, including the system registers, are located in the peripheral bit-banded space of the Cortex-M3 processor memory map. Therefore, bit manipulations may be performed on registers using bit-banded instructions from the Cortex-M3 processor instruction set. These guarantee atomic read-modify-write accesses, which are of use if multiple masters may be accessing a particular location.

7.1.4.9 Fabric Memory Map

There are six regions of 256 KB each, which may be allocated to either FIC_0 or FIC_1 (fabric interrupt controller). This allows large memory mapped windows into the FPGA fabric.

7.1.4.10 Firmware Considerations

The following considerations should be taken into account while implementing Fabric logic:

- **Configuring the AHB bus matrix:** For the mode changes (change of protection region, memory map mode, programmable weights and programmable maximum latency), user firmware should take care that all the masters are in IDLE STATE (where no data transfer is required) for a sufficient amount of time—10 IDLE cycles (ten clock cycles)—before and after the mode change.
- **HBURST support in eNVM slave:** HBURST is supported for an IC bus master to eNVM slaves only. SW_WEIGHT_IC of MASTER_WEIGHT0_CR is configured such that the value of SW_WEIGHT_IC is equal to or greater than the number of bursts. For example, for a burst of 8, the SW_WEIGHT_IC should be at least 8 or greater than 8.
- Avoid using infinite firmware loops in eSRAM which result in preventing WRR masters from accessing the eSRAM. A typical example would be a tight polling loop in the Cortex-M3 processor firmware, executing code from eSRAM, which is polling a location in the same eSRAM and consuming its full bandwidth, thereby not allowing a lower-priority master (such as Ethernet MAC) to access the eSRAM in order to perform the write of the data for which the polling loop is waiting. This leads to a hung system.
This is due to the use of the fixed priority for processor masters in the arbitration algorithm and the possibility of eSRAM being used for both instruction fetches (I and D busses) and data accesses

(SBus). Microsemi recommends that the Cortex-M3 processor firmware is stored in a separate eSRAM from the data storage of other services.

7.1.4.11 Memory Security

After reset, all master ports on the AHB bus matrix are enabled. There are separate user-defined flash configuration bits that control read and write access for each memory slave from various masters, which are organized in groups. The pairing of the masters and the slaves with respect to the bits set in the security registers are given in detail in the following table. Read access and write access can be independently controlled by separate read and write flash bits. The detailed bit configuration of these S registers is given in the [System Register Block](#), page 670.

Table 146 • Pairing of Masters and Slaves

SYSREG Register	Masters	Slaves				
		MS0	MS1	MS2	MS3	MS6
		eSRAM 0	eSRAM1	eNVM_0	eNVM_1	MSS DDR Bridge
MM0_1_2_SECURITY	MM0: M3 DCode bus, cache	RW	RW	RW	RW	RW
	MM1: M3 ICode bus	RW	RW	RW	RW	RW
	MM2: M3 system bus	RW	RW	RW	RW	RW
MM4_5_FIC64_SECURITY	MM4: FIC_0	RW	RW	RW	RW	RW
	MM5: FIC_1	RW	RW	RW	RW	RW
	DDR_FIC	RW	RW	RW	RW	RW
MM3_6_7_8_SECURITY	MM3: HPDMA	RW	RW	RW	RW	RW
	MM6: MAC	RW	RW	RW	RW	RW
	MM7: PDMA	RW	RW	RW	RW	RW
	MM8: USB	RW	RW	RW	RW	RW
MM9_SECURITY	MM9: System controller	RW	RW	RW	RW	RW

An access attempt by a master where the corresponding master port is blocked (by a flash configuration bit setting) causes the AHB bus matrix to assert HRESP to the master and terminate the transaction. If a blocked port is attempting a read access, the read data is returned as garbage. If the blocked port is attempting a write, the write of data does not occur to any slave. In both cases, one of the SW_ERRORSTATUS bits is asserted. DDR_FIC is not part of the AHB bus matrix but can be blocked from accessing the MSS DDR (MDDR) subsystem.

7.2 How to Use AHB Bus Matrix

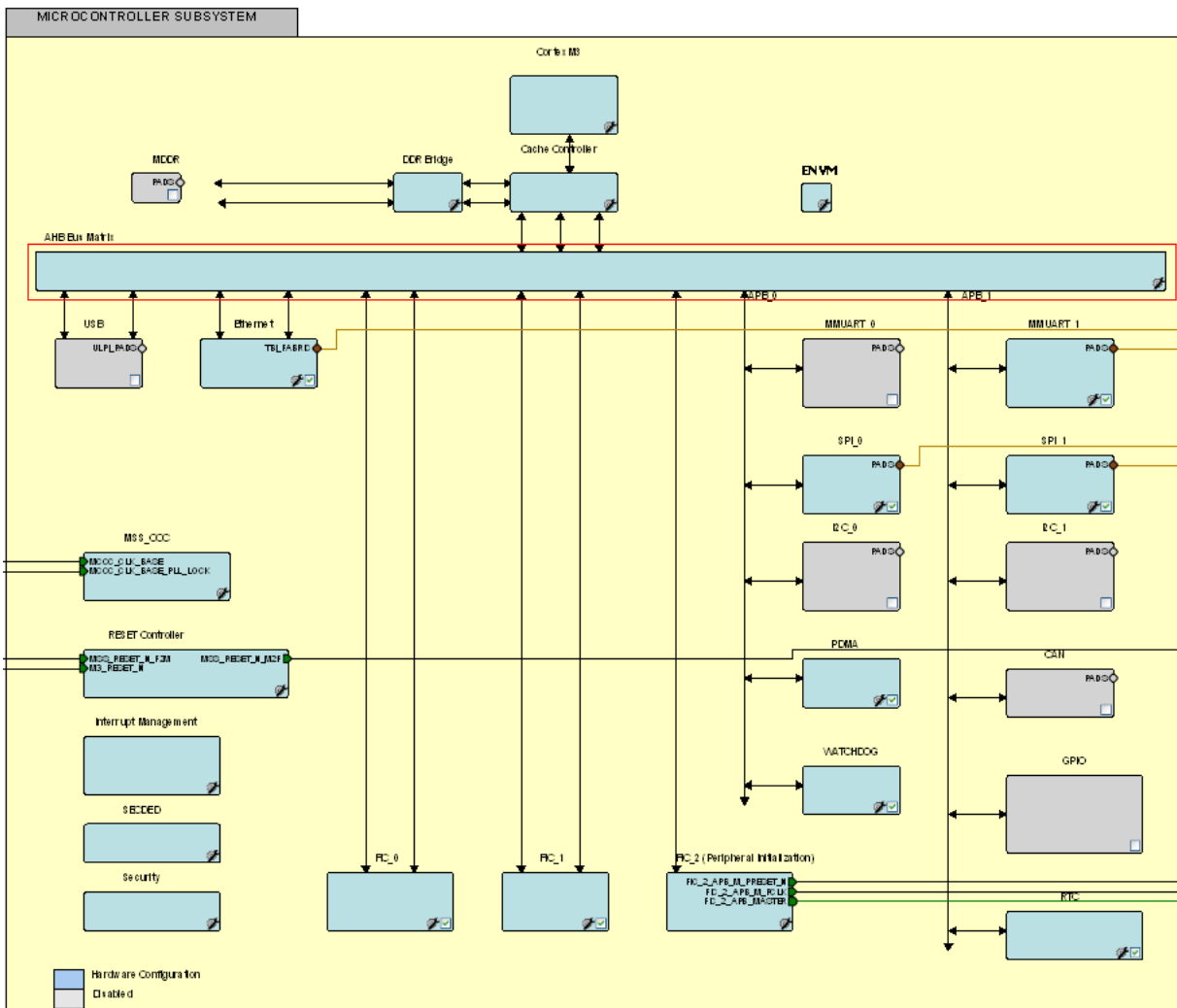
This section describes how to use the AHB bus matrix in an application.

7.2.1 Design Flow

The following steps are used to configure the AHB bus matrix in the application:

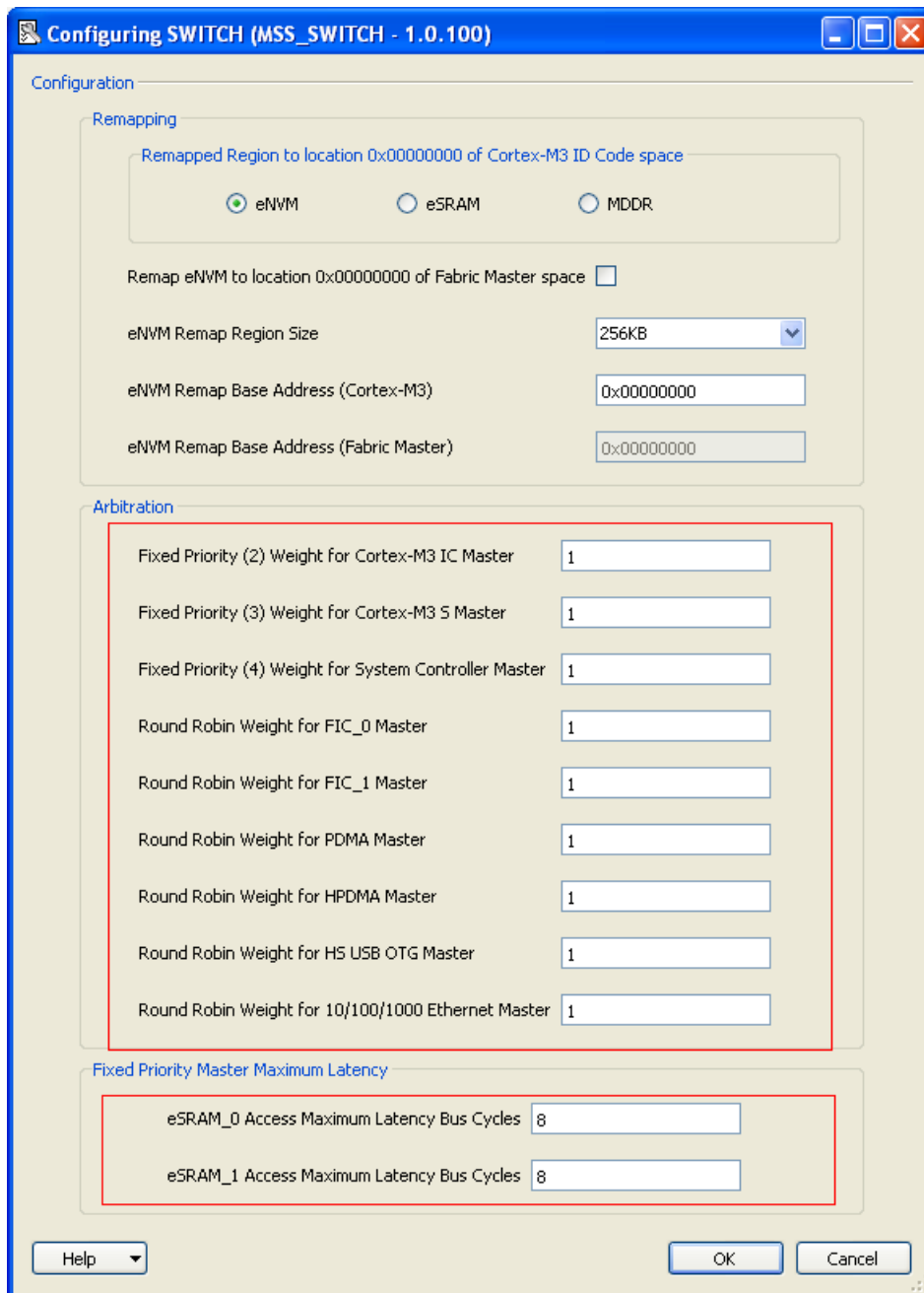
1. Configure the AHB bus matrix by using the MSS configurator in the application, as shown in the following figure.

Figure 118 • AHB Bus Matrix in Libero SoC Design MSS Configurator



2. Click on **AHB Bus Matrix** to configure it. The AHB Bus Matrix configuration window is shown in the following figure.
 - The AHB bus matrix provides support to remap eNVM, eSRAM and DDR memory regions to location 0x00000000 of the Cortex-M3 ID code space. It also provides an option to remap eNVM for a Soft Processor eNVM Remap. To enable remapping for eNVM, eSRAM and DDR select appropriate option from remapping section of AHB Bus Matrix configurator.

Figure 119 • AHB Bus Matrix Configuration Window



- Enter the **weight values** for the masters in arbitration section, to configure the programmable weight registers MASTER_WEIGHT0_CR and MASTER_WEIGHT1_CR. These are located in the SYSREG block with the required weight values. The weight values range is from 1 to 32.
- Enter the **maximum latency values** for the fixed priority masters to configure ESRAM_MAX_LAT registers that are located in the SYSREG block. This decides the peak wait time for a fixed priority master arbitrating for eSRAM access while

the WRR master is accessing the slave. Slave maximum latency can be configured from 1 to 8 clock cycles (8 by default).

Note: ESRAM_MAX_LAT is only supported for fixed priority masters addressing eSRAM slaves. It has no effect on WRR masters.

3. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component**. For more information on generation of the component, refer to the [Libero SoC User Guide](#).
4. Click **Generate Bitstream** under **Program Design** to complete *.fdb file generation.

Note: The MSS AHB Bus Matrix supports full behavioral simulation models. Refer to the [SmartFusion2 MSS BFM Simulation User Guide](#) for information.

7.3 Register Map

The following table lists the AHB bus matrix control registers in the SYSREG block.

Table 147 • AHB Bus Matrix Register Map

Register Name	Register Type	Flash Write Protect	Reset Source	Description
MASTER_WEIGHT0_CR	RW-P	Register	SYSRESET_N	Configures WRR master arbitration scheme for masters.
MASTER_WEIGHT1_CR	RW-P	Register	SYSRESET_N	Configures WRR master arbitration scheme for masters.
MM0_1_2_SECURITY	RO-U	N/A	SYSRESET_N	Security bits for masters 0, 1, and 2
MM4_5_DDR_FIC_SECURITY/MM4_5_FIC64_SECURITY	RO-U	N/A	SYSRESET_N	Security bits for masters 4, 5, and DDR_FIC
MM3_6_7_8_SECURITY	RO-U	N/A	SYSRESET_N	Security bits for masters 3, 6, 7, and 8
MM9_SECURITY	RO-U	N/A	SYSRESET_N	Security bits for master 9
MSS_EXTERNAL_SR	SW1C	N/A	SYSRESET_N	AHB bus matrix error status. Writing a 1 clears the status.
ESRAM_CR	RW-P	Register	SYSRESET_N	This register configures eSRAM.
ENVN_CR	RW-P	Register	SYSRESET_N	This register configures eNVM parameters.
ESRAM_MAX_LAT	RW-P	Register	SYSRESET_N	This register configures maximum latency for accessing eSRAM0/1 slave.
ENVN_REMAP_BASE_CR	RW-P	Register	SYSRESET_N	This signal indicates the base address of the segment in eNVM which is to be remapped to location 0H.
ENVN_REMAP_FAB_CR	RW-P	Register	SYSRESET_N	Configures where eNVM is mapped in fabric master space.
DDRNB_NB_ADDR_CR	RW-P	Register	SYSRESET_N	This register indicates the base address of the non-bufferable address region.
DDRNB_NB_SIZE_CR	RW-P	Register	SYSRESET_N	This register indicates the size of the non-bufferable address region.
DDR_CR	RW-P	Register	SYSRESET_N	This register configures DDR parameters.

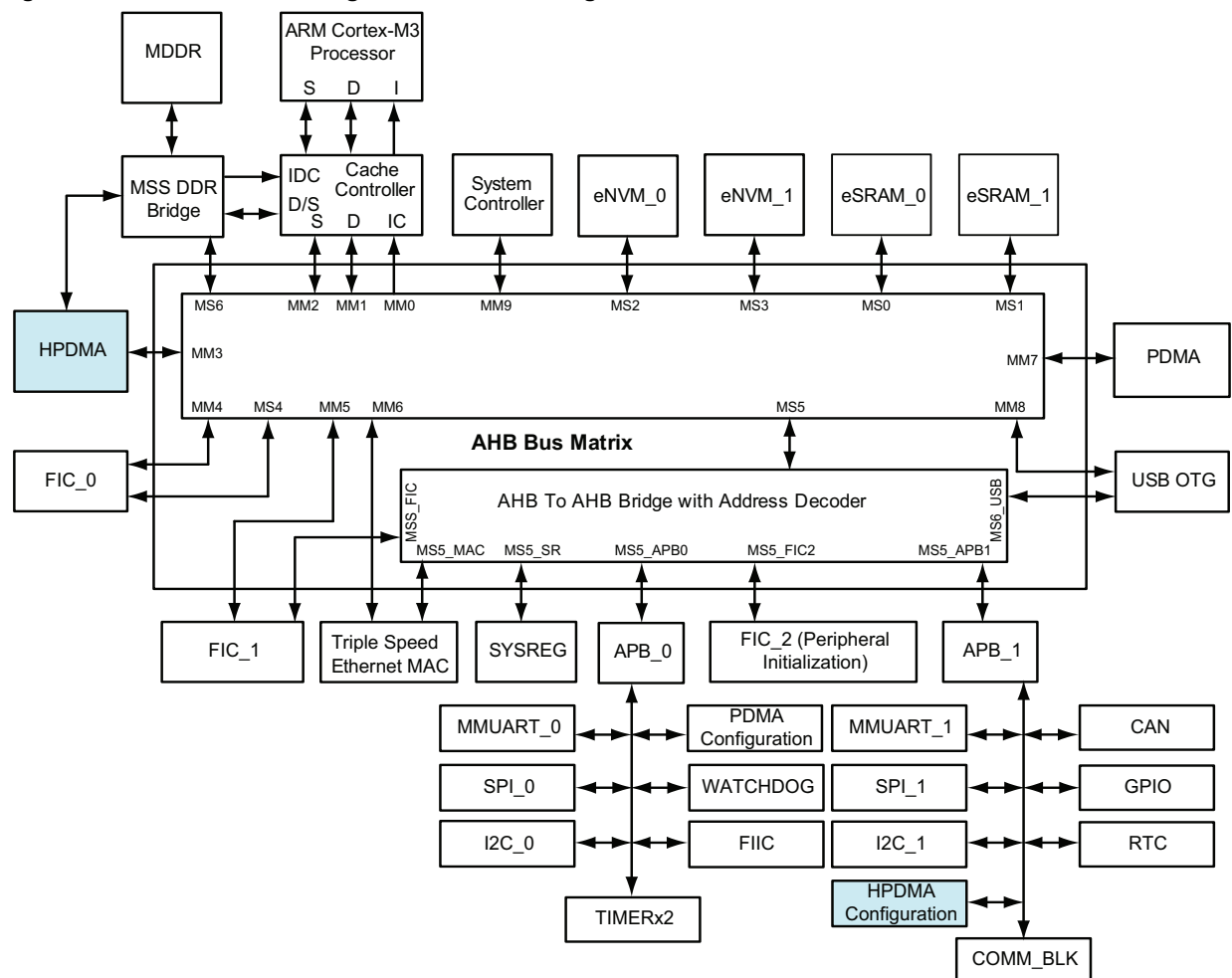
8 High Performance DMA Controller

The high performance DMA Controller (HPDMA) provides fast data transfer between the MSS DDR bridge and MSS memories. The MSS memories are eSRAM0, eSRAM1, eNVM0, and eNVM1. The DDR bridge connects to External DDR memory.

The following figure shows HPDMA interfacing with AHB Bus Matrix and MSS DDR bridge. AHB bus masters such as the Cortex-M3 processor can offload the high speed memory transfers to HPDMA, making the master available for performing other tasks. All transfers by the HPDMA are full word transfers. The HPDMA controller has two AHB masters, MSS DDR Bridge and AHB bus matrix master (MM0-MM9) which functions concurrently to enable high performance data transfers. The configuration of HPDMA is done through the APB interface.

One of the main applications for which the HPDMA can be used is paging access by the processor. The main data is stored in a large DDR space and relevant chunks of this data would be transferred as needed via the HPDMA to the eSRAM, where it can be processed faster.

Figure 120 • HPDMA Interfacing With MSSDDR Bridge and AHB Bus Matrix



8.1 Features

- Faster read/write operations with two concurrent AHB masters
- 32-bit AHB operation at 200 MHz
- 32-bit APB slave interface for control and status registers at 25/50/100/200 MHz
- Internal 32-bit control, status, and debug registers
- Single DMA channel with four queuing HPDMA descriptors, serviced with round robin priority
- Up to 64 KB data transfer in single channel request
- 32-byte internal data buffer
- Supports word aligned data transfers
- Interrupts for DMA transfer complete and transfer errors
- DMA transfer pause
- Individual descriptor reset
- Data transfer in little-endian format

8.2 Functional Description

HPDMA has a single channel which can process up to four service requests (HPDMA descriptor) in a round robin fashion. To process each request, HPDMA descriptor is configured by an AHB bus matrix master through APB interface. The AHB bus matrix master can be Cortex-M3, USB, Ethernet and Fabric master. The HPDMA APB interface is connected on APB_1, which is an AHB to APB bridge as shown in the preceding figure. HPDMA then reads data from the source memory and transfers data to the destination.

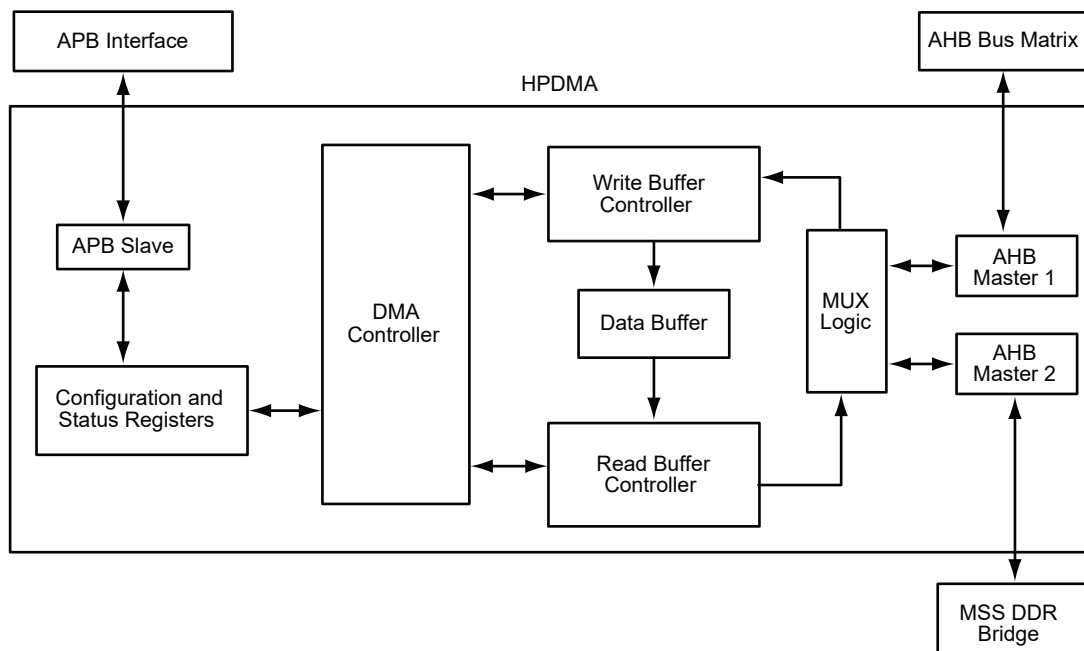
This section provides the detailed description of the HPDMA.

Architecture Overview

HPDMA mainly consists of following sub-blocks, as shown in the following figure:

- Interfaces
- Configuration and Status Registers
- DMA Controller
- Write Buffer Controller
- Read Buffer Controller
- Data Buffer

Figure 121 • HPDMA Controller Block Diagram



8.2.0.1 Interfaces

There are two types of interfaces used for communicating with HPDMA:

- 32-bit APB slave interface for configuration
- Two AHB master interfaces (AHB-M1, AHB-M2) for data transfers:
 - AHB Master 1 does the read/write transfers at the AHB bus matrix end
 - AHB Master 2 does the read/write transfers at the MSS DDR bridge end

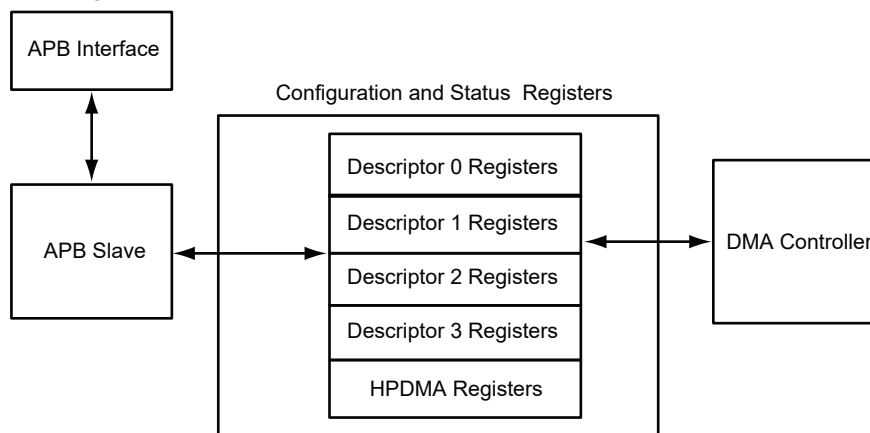
8.2.0.2 Configuration and Status Registers

The configuration and status registers of the HPDMA controller are accessed through a 32-bit APB slave, as shown in the following figure. To enable and use HPDMA services, the AHB bus matrix master must configure the 32-bit wide descriptor registers.

There are four descriptors available with the HPDMA controller. Each descriptor has the following five registers:

- Source memory address register
- Destination memory address register
- Control register
- Status register
- Pending Transfer register

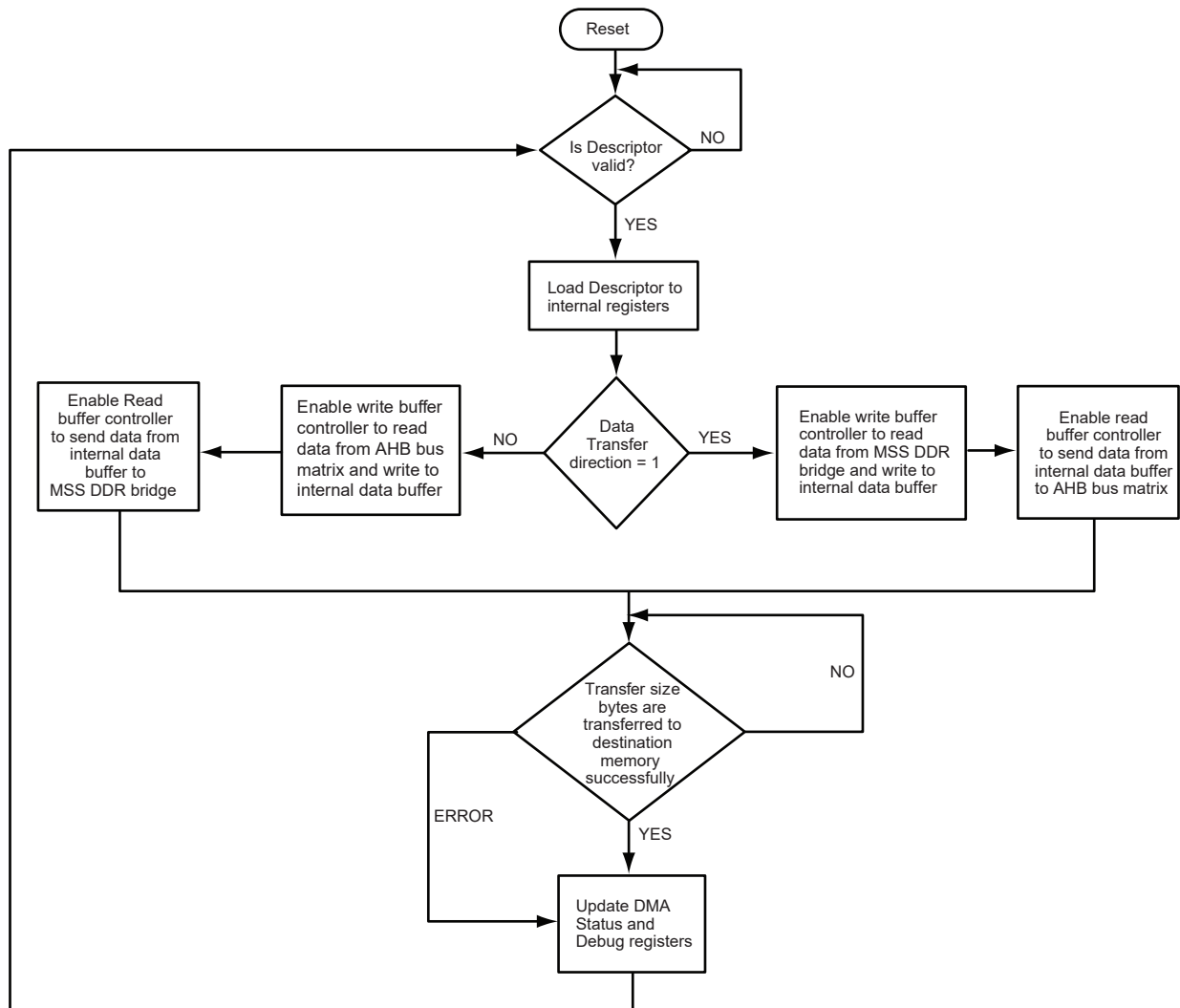
Figure 122 • HPDMA Registers



8.2.0.3 DMA Controller

The DMA controller controls and monitors transactions on the source and destination AHB master interfaces. When a descriptor is configured, the DMA controller enables the write buffer controller to read data from the appropriate source memory (AHB bus matrix or MSS DDR bridge) and transfer it into the internal data buffer. In a similar way, the DMA controller enables read buffer controller to read the data from the internal data buffer and transfers it to the destination memory. The following figure shows the detailed DMA Controller flow.

Figure 123 • DMA Controller Flow Chart



8.2.0.4 Write Buffer Controller

The write buffer controller enables the appropriate AHB master (AHB-M1 or AHB-M2) to read the data from source memory. To initiate read transfers on the AHB bus, the write buffer controller provides the read address and asserts the ready signal. The AHB master acknowledges, and the write buffer controller writes the source memory data to the internal data buffer.

If the data buffer is full, the write buffer controller initiates idle transfers on the AHB bus, and asserts ready signal when at least one data buffer is available. The write buffer controller pauses the DMA transfers when the descriptor pause bit is enabled, and resumes the transfers as soon as the pause bit is disabled. When the last count value is reached, the AHB slave acknowledges the last transfer.

8.2.0.5 Read Buffer Controller

The read buffer controller places the address and asserts the ready signal to the AHB master (AHB-M1 or AHB-M2). Depending on the transfer direction, AHB-M1 or AHB-M2 initiates the data transfers from internal data buffer to destination memory.

If the data buffer is empty or if the DMA controller pause bit is enabled, then the read buffer controller initiates IDLE transfers on the AHB bus.

8.2.0.6 Data Buffer

The data buffer block is 32 bits wide and 8 words deep. Data buffer read/write operations are performed on the rising edge of the clock signal. There are 4-bit read and write pointers that increment on read and write.

The 3 least significant bits (LSBs) are used to address the 8 locations; the most significant bit (MSB) of the read and write pointers is used to signal the data buffer empty and full.

8.2.0.6.1 Data Buffer Full and Empty

When the read pointer and write pointer are equal, the data buffer is empty. When the 3 LSBs of read pointer and write pointer are equal and the MSBs of the read pointer and write pointer are not equal, the data buffer is full.

8.2.1 Initialization

To initiate and setup DMA transactions, HPDMA has to be initialized. The initialization process starts with a reset sequence followed by Channel configuration and interrupt configuration.

8.2.1.1 Reset

The HPDMA registers are reset on power-up. The HPDMA can be reset by configuring the Bit 17 of SOFT_RESET_CR system register.

8.2.1.2 Descriptor Configuration

Before configuring each HPDMA channel, the round robin weight is specified if needed, using the MASTER_WEIGHT_CR register or configuring the AHB bus matrix in Libero SoC.

To configure each HPDMA descriptor, the following registers have to be set:

- Descriptor Control registers:
 - Direction: bit 1 of HPDMADXCXCR_REG (where X is 0 to 3)
 - Transfer size in bytes: bits[15:0] of HPDMADXCXCR_REG (where X is 0 to 3)
 - Enable Interrupts: bits[22:20] of HPDMADXCXCR_REG (where X is 0 to 3)
- 32 bit Source memory start Address: bits[31:0] of HPDMADXSAR_REG (where X is 0 to 3)
- 32 bit Destination memory start Address: bits[31:0] of HPDMADXDAR_REG (where X is 0 to 3)

8.2.1.3 Interrupt

There are two interrupts: HPD_XFR_CMP_INT and HPD_XFR_ERR_INT from the HPDMA to the NVIC on the Cortex-M3 processor. The interrupt signals are mapped to one of the IRQs in the Cortex-M3 NVIC controller.

The interrupt signals are also mapped to the dedicated interrupt signal MSS_INT_M2F[9] and the MSS_INT_M2F[22] of the fabric interface interrupt controller (FIIC).

This is to interrupt the user logic instantiated in the FPGA fabric. To enable HPDMA interrupts, the 9th bit (HPD_XFR_CMP_INT_EN) and the 22nd bit (HPD_XFR_ERR_INT_EN) of INTERRUPT_ENABLE0 register (located at address 0x40006000) has to be set. The status of the interrupts to FIIC can be determined by reading the 9th and 22nd bits of the INTERRUPT_REASON0 register (located at 0x40006008).

To determine the descriptor transfer status, monitor the Descriptor status register (HPDMADXSAR, where X is 0 to 3). Before start of transaction, the enabled Descriptor interrupt bits are to be cleared. Refer to HPDMAICR_REG for clearing of interrupts.

8.2.2 Details of Operation

After initialization, the HPDMA is ready to function in one of the two following data transfer modes:

- AHB bus matrix to MSS DDR bridge
- MSS DDR bridge to AHB bus matrix

For initiation of the above data transfer modes, a descriptor valid bit has to be set (that is, bit 16 of the Descriptor control register is asserted). If all the four descriptors are configured and set to valid, the descriptor transfer begins and executes in a round robin fashion. If any of the descriptor is paused by setting the bit 19 of Descriptor control register, the HPDMA stops the data transfer. HPDMA resumes the operation once the pause bit is reset. The pending transfers of the source and destination can be read from the Descriptor pending transfer register (HPDMADXPTR, where X is 0 to 3).

HPDMA can service the next descriptor only after the pending transfer of the current descriptor is complete. The data transfer completion interrupt is monitored using bit 20 of the Descriptor control register and bit 1 of the Descriptor status register. Refer to the [HPDMA Register Bit Definitions](#), page 247 for more information on HPDMA registers.

8.3 How to Use HPDMA

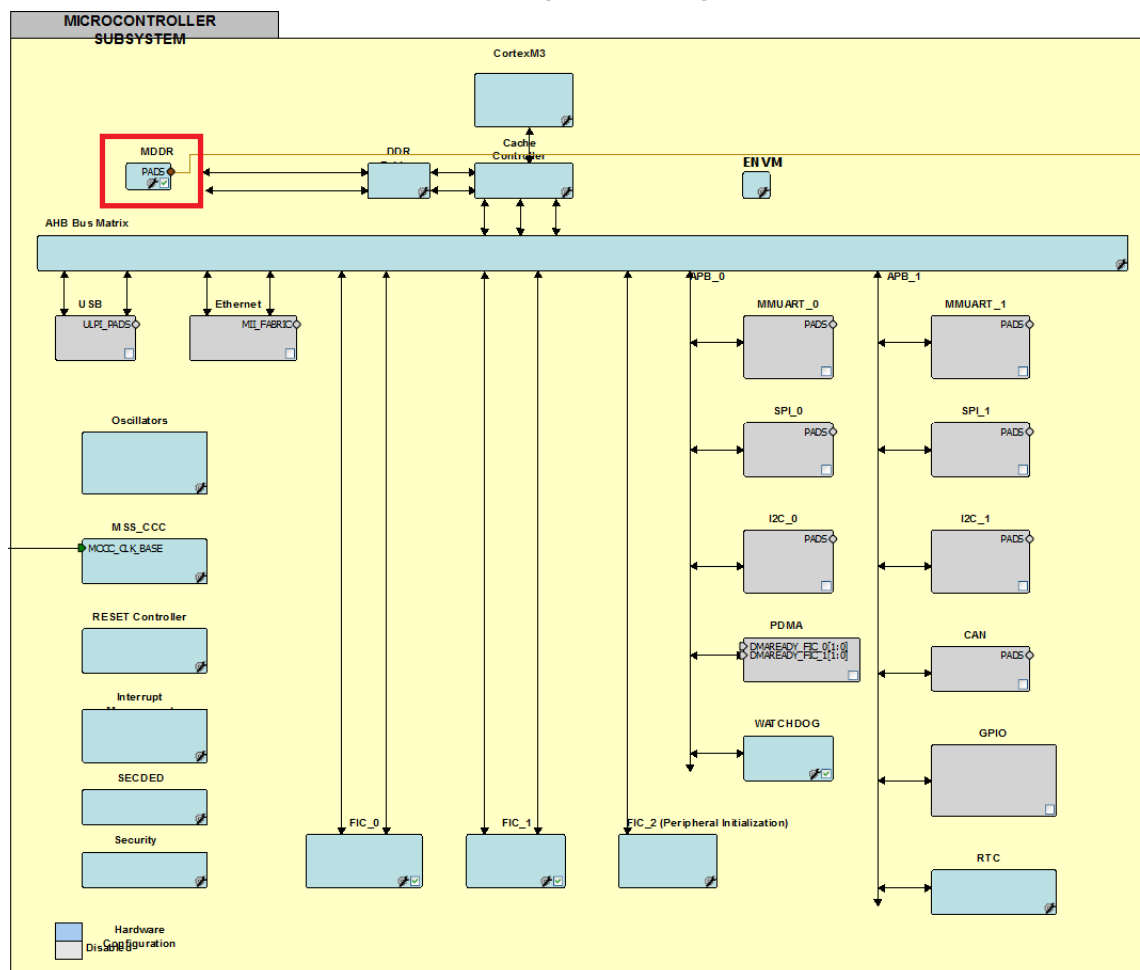
This section describes how to use the HPDMA in an application.

8.3.1 Design Flow

The following steps describe how to enable HPDMA in the application:

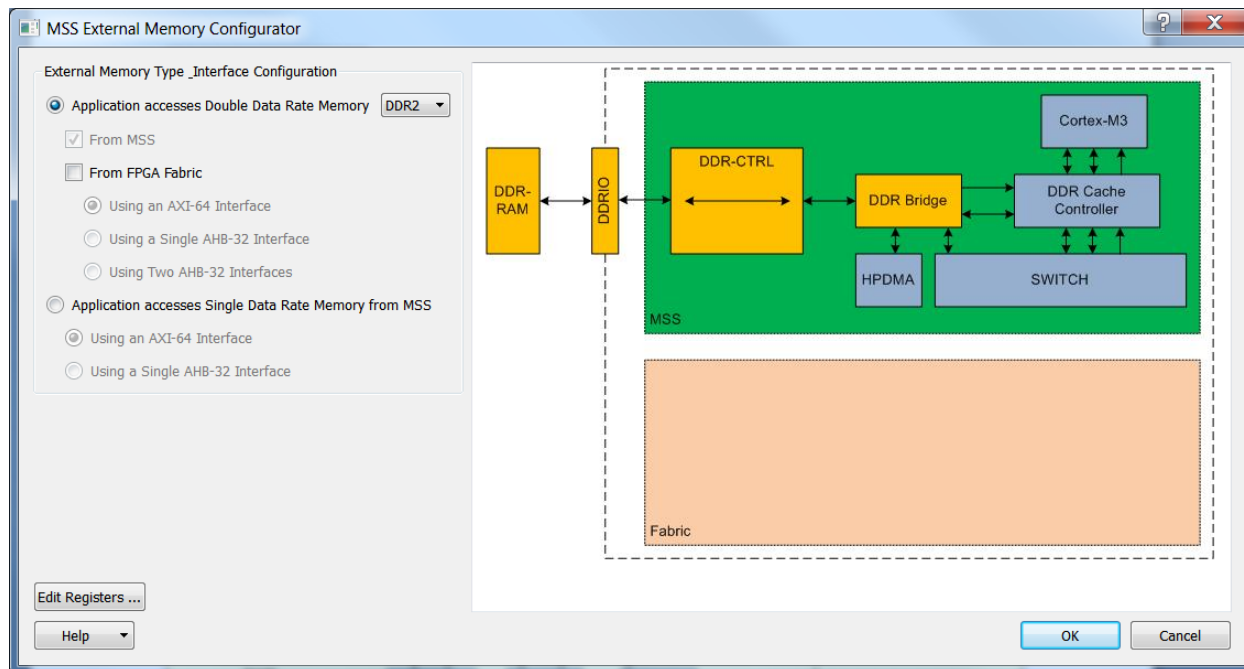
1. Enable HPDMA by using MDDR in the application, as shown in the following figure.

Figure 124 • Enable HPDMA in the Libero SOC Design MSS Configurator



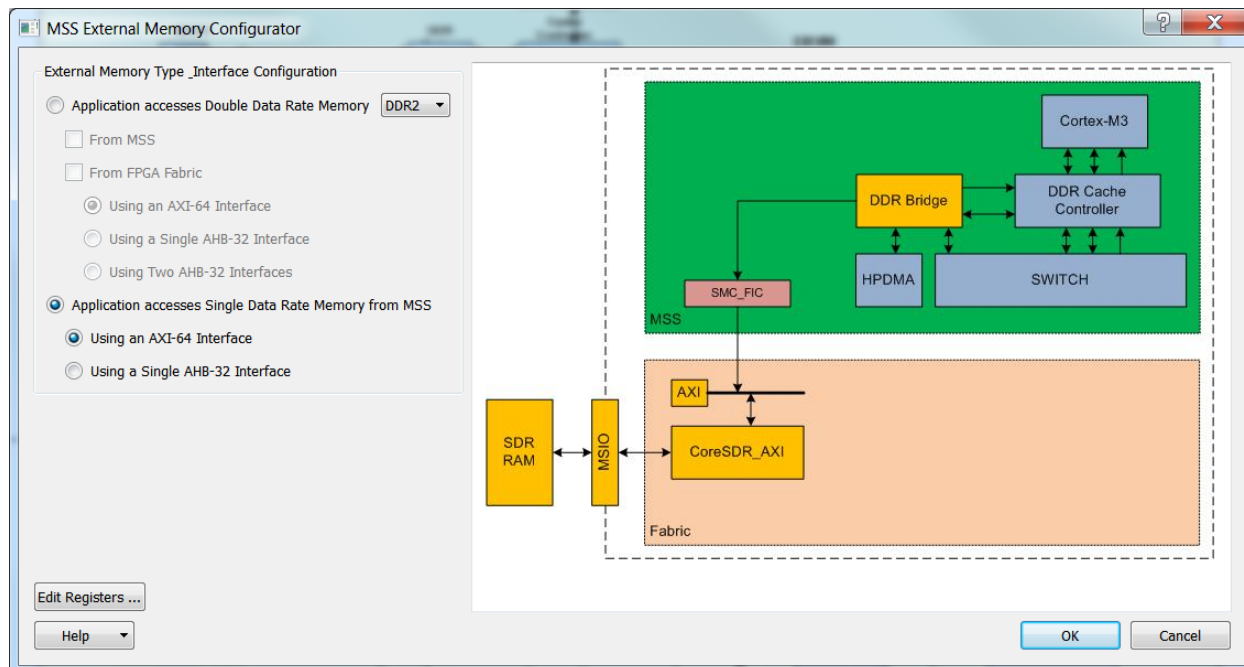
- To configure the HPDMA to transfer data between DDR memory and MSS internal memory, make the selection in the MSS external memory configurator as shown in the following figure.

Figure 125 • HPDMA Transfers Data Between DDR Memory and MSS Internal Memory



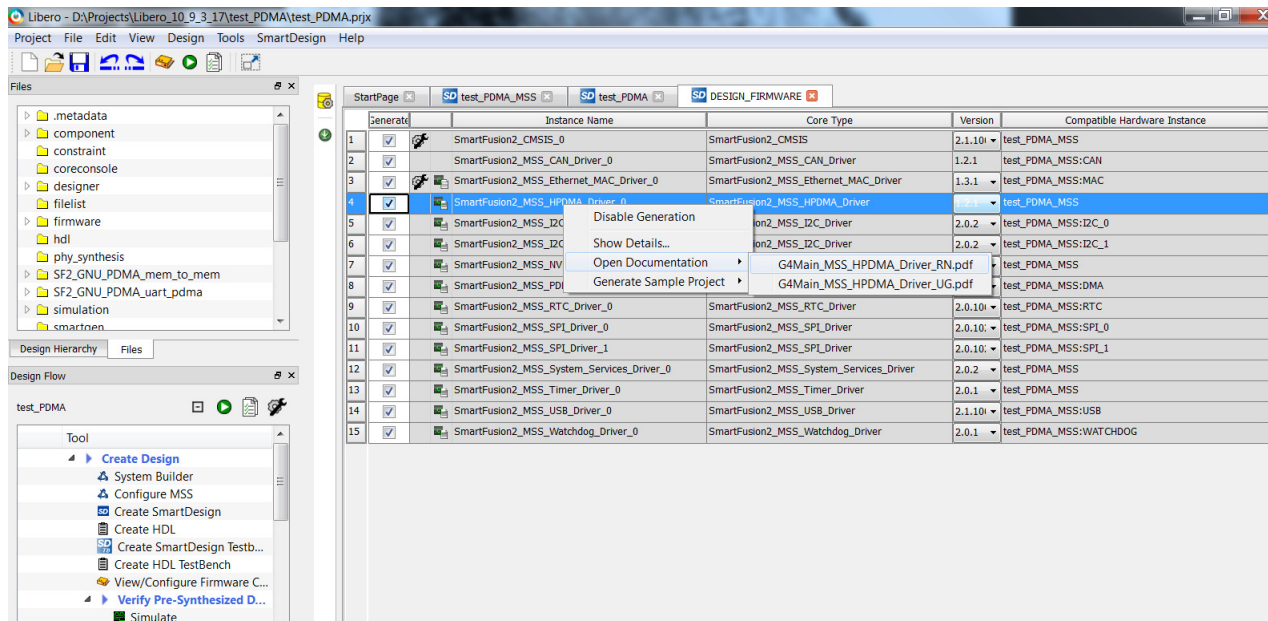
- To configure the HPDMA to transfer data between SDR memory and MSS internal memory, make the selection in the MSS external memory configurator as shown in the following figure.

Figure 126 • HPDMA Transfers Data Between SDR Memory and MSS Internal Memory



4. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component** from the menu. The firmware driver folder and SoftConsole workspace is included in the project. For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace is included in the project. Click the highlighted **Configure firmware** button as shown in the following figure to find the RTC drivers.

Figure 127 • HPDMA Driver User Guide



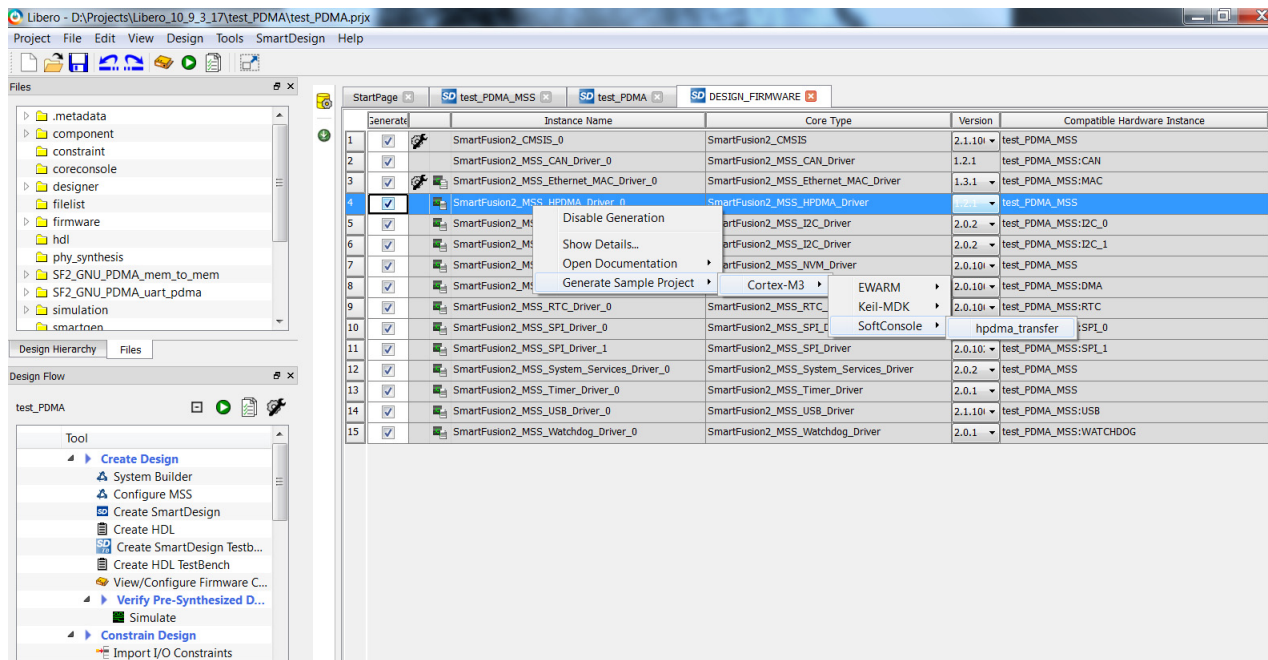
5. Click **Generate Bitstream** under **Program Design** to complete *.fdb file generation.
6. Double-click **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_hpdma firmware driver (mss_hpdma.c and mss_hpdmac.h), which provides a set of functions for controlling the MSS HPDMA transfers. The mss_hpdma firmware driver can also be downloaded from the Microsemi firmware catalog. The following table shows the list of APIs for HPDMA. For more information on the APIs, refer to the **SmartFusion2_MSS_HPDMAC_Driver_UG** shown in the preceding figure.

Table 148 • MSS HPDMA APIs

Category	API	Description and Usage
Initialization	MSS_HPDMAC_init()	Initializes HPDMA
Control of descriptor transfer	MSS_HPDMAC_start()	Starts HPDMA transfer
	MSS_HPDMAC_pause()	Pauses HPDMA transfer
	MSS_HPDMAC_resume()	Resumes HPDMA transfer
	MSS_HPDMAC_abort()	Aborts the HPDMA transfer
Status of current transfer	MSS_HPDMAC_get_pending_counters()	Gets the number of pending transfers in bytes
	MSS_HPDMAC_get_transfer_state()	Gets the transfer state when transfer in progress
Interrupt control functions	MSS_HPDMAC_enable_irq()	Enables either transfer interrupt or error interrupt
	MSS_HPDMAC_disable_irq()	Disables either transfer interrupt or error interrupt
	MSS_HPDMAC_clear_irq()	Clears either transfer interrupt or error interrupt

- For more information on HPDMA usage, the sample projects are available and can be generated, as shown in the following figure.

Figure 128 • HPDMA Examples



8.3.2 HPDMA Use Models

This section explains the use models and gives directions for using HPDMA in an application. The SmartFusion2 soft memory controller fabric interface controller (SMC_FIC) is used to access external bulk memories other than DDR through the FPGA fabric. The SMC_FIC can be used with a soft memory controller for the MSS to access memories such as SDRAM, flash, and SRAM. MSS masters communicate with the SMC_FIC through an MSS DDR bridge present in the MSS. For more information on SMC_FIC, refer to the "Soft Memory Controller Fabric Interface Controller" chapter in the *UG0446: SmartFusion2 and IGLOO2 FPGA High Speed DDR Interfaces User Guide*.

8.3.2.1 Use Model 1: AHB Bus Matrix to MSS DDR Bridge/MSS DDR Bridge to AHB Bus Matrix

- Enable HPDMA by using MDDR or configuring switch and MSSDDR bridge in Libero SoC.
- Initialize the HPDMA using `MSS_HPDMADriver_init()`.
- Check HPDMAEDR register bits using `MSS_HPDMADriver_get_transfer_state()`.
- Check Clear if any interrupts are pending `MSS_HPDMADriver_clear_irq()`.
- Configure the descriptor transfer size, source address, destination address, transfer direction and enable the valid HPDMA descriptor register using `MSS_HPDMADriver_start()`.
- Check HPDMAEDR register bits using `MSS_HPDMADriver_get_transfer_state()`.
- To pause the data transfers use `MSS_HPDMADriver_pause()`.
- Get the pending transfer bytes using `MSS_HPDMADriver_get_pending_counters()`.
- To resume the data transfers use `MSS_HPDMADriver_pause()`.
- To check the status of the transfer use `MSS_HPDMADriver_get_transfer_state()`.

8.4 HPDMA Controller Register Map

The following table summarizes the HPDMA controller register map. The sections that follow detail register bit descriptions of status, configuration, and debug registers. All the register bits are active high; on reset they assume default values. Register R/W corresponds to external processor accessibility. The address range of the HPDMA APB registers is x40014000 to x40014FFF. Only the 7 LSBs are considered for addressing the registers.

Table 149 • HPDMA Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
HPDMAEDR_REG	x00	R	x0F	HPDMA Empty Descriptor register
HPDMAD0SAR_REG	x04	R/W	x00	Descriptor 0 source memory start address
HPDMAD0DAR_REG	x08	R/W	x00	Descriptor 0 destination memory start address
HPDMAD0SR_REG	x0C	R/W	x00	Descriptor 0 Control register
HPDMAD0SR_REG	x10	R	x00	Descriptor 0 Status register
HPDMAD0PTR_REG	x14	R	x00	Descriptor 0 Pending Transfer register
HPDMAD1SAR_REG	x18	R/W	x00	Descriptor 1 source memory start address
HPDMAD1DAR_REG	x1C	R/W	x00	Descriptor 1 destination memory start address.
HPDMAD1CR_REG	x20	R/W	x00	Descriptor 1 Control register
HPDMAD1SR_REG	x24	R	x00	Descriptor 1 Status register
HPDMAD1PTR_REG	x28	R	x00	Descriptor 1 Pending Transfer register
HPDMAD2SAR_REG	x2C	R/W	x00	Descriptor 2 source memory start address
HPDMAD2DAR_REG	x30	R/W	x00	Descriptor 2 destination memory start address
HPDMAD2CR_REG	x34	R/W	x00	Descriptor 2 Control register
HPDMAD2SR_REG	x38	R	x00	Descriptor 2 Status register
HPDMAD2PTR_REG	x3C	R	x00	Descriptor 2 Pending Transfer register
HPDMAD3SAR_REG	x40	R/W	x00	Descriptor 3 source memory start address
HPDMAD3DAR_REG	x44	R/W	x00	Descriptor 3 destination memory start address
HPDMAD3CR_REG	x48	R/W	x00	Descriptor 3 Control register
HPDMAD3SR_REG	x4C	R	x00	Descriptor 3 Status register
HPDMAD3PTR_REG	x50	R	x00	Descriptor 3 Pending Transfer register
HPDMAICR_REG	x54	W	x00	HPDMA Interrupt Clear register
HPDMADR_REG	x58	R	x01	HPDMA Debug register

8.4.1 HPDMA Register Bit Definitions

8.4.1.1 HPDMA Empty Descriptor Register

Table 150 • HPDMAEDR_REG

Bit Number	Name	Reset Value	Description
0	HPDMAEDR_DCP_EMPTY[0]	1	Descriptor 0 is empty and ready for software configuration. 1: Descriptor 0 is empty and ready to configure. 0: Descriptor 0 is already configured and descriptor transfer is in progress/queue. At the end of the descriptor transfer, either on transfer error or transfer done, the HPDMA controller asserts this bit High.
1	HPDMAEDR_DCP_EMPTY[1]	1	Descriptor 1 is empty and ready for software configuration. 1: Descriptor 1 is empty and ready to configure. 0: Descriptor 1 is already configured and descriptor transfer is in progress/queue. At the end of the descriptor transfer, either on transfer error or transfer done, the HPDMA controller asserts this bit High.
2	HPDMAEDR_DCP_EMPTY[2]	1	Descriptor 2 is empty and ready for software configuration. 1: Descriptor 2 is empty and ready to configure. 0: Descriptor 2 is already configured and descriptor transfer is in progress/queue. At the end of the descriptor transfer, either on transfer error or transfer done, the HPDMA controller asserts this bit High.
3	HPDMAEDR_DCP_EMPTY[3]	1	Descriptor 3 is empty and ready for software configuration. 1: Descriptor 3 is empty and ready to configure. 0: Descriptor 3 is already configured and descriptor transfer is in progress/queue. At the end of the descriptor transfer, either on transfer error or transfer done, the HPDMA controller asserts this bit High.
4	HPDMAEDR_DCP_CMPLT[0]	0	Descriptor 0 transfer complete. 1: Descriptor 0 transfer completed successfully. 0: Descriptor 0 transfer not completed. When the descriptor 0 transfer is completed, either with transfer error or transfer done, HPDMA controller asserts this bit High. This bit is cleared on writing '1' to the HPDMAICR_CLR_XFR_INT[0] bit of the HPDMA Interrupt Clear register or when the HPDMACR_DCP_VALID[0] bit of descriptor 0 Control register is set.

Table 150 • HPDMAEDR_REG (continued)

Bit Number	Name	Reset Value	Description
5	HPDMAEDR_DCP_CMPLT[1]	0	Descriptor 1 transfer complete. 1: Descriptor 1 transfer completed successfully. 0: Descriptor 1 transfer not completed. When the descriptor 1 transfer is completed, either with transfer error or transfer done, the HPDMA controller asserts this bit High. This bit is cleared on writing '1' to the HPDMAICR_CLR_XFR_INT[1] bit of the HPDMA Interrupt Clear register or when the HPDMACR_DCP_VALID[1] bit of the descriptor 1 Control register is set.
6	HPDMAEDR_DCP_CMPLT[2]	0	Descriptor 2 transfer complete. 1: Descriptor 2 transfer completed successfully 0: Descriptor 2 transfer not completed When the descriptor 2 transfer is completed, either with transfer error or transfer done, the HPDMA controller asserts this bit High. This bit is cleared on writing '1' to the HPDMAICR_CLR_XFR_INT[2] bit of the HPDMA Interrupt Clear register or when the HPDMACR_DCP_VALID[2] bit of the descriptor 2 Control register is set.
7	HPDMAEDR_DCP_CMPLT[3]	0	Descriptor 3 transfer complete. 1: Descriptor 3 transfer completed successfully 0: Descriptor 3 transfer not completed When the descriptor 3 transfer is completed, either with transfer error or transfer done, the HPDMA controller asserts this bit High. This bit is cleared on writing '1' to the HPDMAICR_CLR_XFR_INT[3] bit of the HPDMA Interrupt Clear Register or when the HPDMACR_DCP_VALID[3] bit of the descriptor 3 control register is set.
8	HPDMAEDR_DCP_ERR[0]	0	Descriptor 0 transfer error. 1: Descriptor 0 transfer error 0: No descriptor 0 transfer error This bit is asserted High if an error occurs during the descriptor 0 transfer at either source or destination end. This bit is cleared on writing '1' to the HPDMAICR_CLR_XFR_INT[0] bit of the HPDMA Interrupt Clear register or when the HPDMACR_DCP_VALID[0] bit of the descriptor 0 control register is set.

Table 150 • HPDMAEDR_REG (continued)

Bit Number	Name	Reset Value	Description
9	HPDMAEDR_DCP_ERR[1]	0	Descriptor 1 transfer error. 1: Descriptor 1 transfer error 0: No descriptor 1 transfer error This bit is asserted High, if an error occurs during the descriptor 1 transfer at either source or destination end. This bit is cleared on writing '1' to HPDMAICR_CLR_XFR_INT[1] of the HPDMA Interrupt Clear register, or when the HPDMACR_DCP_VALID[1] bit of Descriptor 1 control register is set.
10	HPDMAEDR_DCP_ERR[2]	0	Descriptor 2 transfer error. 1: Descriptor 2 transfer error 0: No descriptor 2 transfer error This bit is asserted High, if an error occurs during the descriptor 2 transfer at either source or destination end. This bit is cleared on writing '1' to the HPDMAICR_CLR_XFR_INT[2] bit of the HPDMA Interrupt Clear register, or when the HPDMACR_DCP_VALID[2] bit of the descriptor 2 control register is set.
11	HPDMAEDR_DCP_ERR[3]	0	Descriptor 3 transfer error. 1: Descriptor 3 transfer error 0: No descriptor 3 transfer error This bit is asserted High, if an error occurs during the descriptor 3 transfer at either source or destination end. This bit is cleared on writing '1' to HPDMAICR_CLR_XFR_INT[3] of the HPDMA Interrupt Clear register, or when the HPDMACR_DCP_VALID[3] bit of the descriptor 3 control register is set.
12	HPDMAEDR_DCP_NON_WORD_ERR[0]	0	Descriptor 0 non-word aligned transfer size error. 1: Descriptor 0 non-word aligned transfer size error 0: No non-word aligned transfer size error This bit is asserted High, if non-word aligned value is configured in descriptor 0 transfer size field. This bit is cleared on writing '1' to HPDMAICR_NON_WORD_INT[0] of the HPDMA Interrupt Clear register, or when the HPDMACR_DCP_VALID[0] bit of the descriptor 0 Control register is set or when the HPDMACR_DCP_CLR[0] bit of the HPDMA Controller register is set. In this case, HPDMA will continue the transfer by ignoring the 2 LSBs of the transfer size field.

Table 150 • HPDMAEDR_REG (continued)

Bit Number	Name	Reset Value	Description
13	HPDMAEDR_DCP_NON_WORD_ERR[1]	0	Descriptor 1 non-word aligned transfer size error. 1: Descriptor 1 non-word aligned transfer size error 0: No non-word aligned transfer size error This bit is asserted High if a non-word aligned value is configured in the descriptor 1 transfer size field. This bit is cleared on writing '1' to HPDMAICR_NON_WORD_INT[1] of the HPDMA Interrupt Clear register, or when the HPDMACR_DCP_VALID[1] bit of the descriptor 1 Control register is set, or when the HPDMACR_DCP_CLR[1] bit of the HPDMA Controller register is set. In this case, HPDMA will continue the transfer by ignoring the 2 LSBs of the transfer size field.
14	HPDMAEDR_DCP_NON_WORD_ERR[2]	0	Descriptor 2 non-word aligned transfer size error. 1: Descriptor 2 non-word aligned transfer size error 0: No non-word aligned transfer size error This bit is asserted High if a non-word aligned value is configured in the descriptor 2 transfer size field. This bit is cleared on writing '1' to HPDMAICR_NON_WORD_INT[2] of the HPDMA Interrupt Clear register, or when the HPDMACR_DCP_VALID[2] bit of the descriptor 2 Control register is set, or when the HPDMACR_DCP_CLR[2] bit of the HPDMA Controller register is set. In this case, HPDMA will continue the transfer by ignoring the 2 LSBs of the transfer size field.
15	HPDMAEDR_DCP_NON_WORD_ERR[3]	0	Descriptor 3 non-word aligned transfer size error. 1: Descriptor 3 non-word aligned transfer size error 0: No non-word aligned transfer size error This bit is asserted High, if a non-word aligned value is configured in the descriptor 3 transfer size field. This bit clears on writing '1' to HPDMAICR_NON_WORD_INT[3] of the HPDMA Interrupt Clear register, or when the HPDMACR_DCP_VALID[3] bit of the descriptor 3 Control register is set, or when the HPDMACR_DCP_CLR[3] bit of the HPDMA Controller register is set. In this case, HPDMA will continue the transfer by ignoring the 2 LSBs of transfer size field.
[31:16]	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.2 Descriptor 0 Source Address Register

Table 151 • HPDMAD0SAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMASAR_DCP0_SRC_ADRS	0x00	Descriptor 0 source end memory start address

8.4.1.3 Descriptor 1 Source Address Register

Table 152 • HPDMAD1SAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMASAR_DCP1_SRC_ADRS	0x00	Descriptor 1 source end memory start address

8.4.1.4 Descriptor 2 Source Address Register

Table 153 • HPDMAD2SAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMASAR_DCP2_SRC_ADRS	0x00	Descriptor 2 source end memory start address

8.4.1.5 Descriptor 3 Source Address Register

Table 154 • HPDMAD3SAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMASAR_DCP3_SRC_ADRS	0x00	Descriptor 3 source end memory start address

8.4.1.5.1 Notes on the Source Address Register (SAR)

- Address is word aligned at the start.
- Address increments on each successful transfer at the source end.
- HPDMA controller starts reading the data from source memory and transfers to destination memory.
- Software can write all 32-bit source address to prevent non-word aligned transfers at the start and 2 LSBs, 1:0, are masked in the hardware.
- The source address is updated when descriptor transfer is in progress.

8.4.1.6 Descriptor 0 Destination Address Register

Table 155 • HPDMAD0DAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMADAR_DCP0_DST_ADRS	0x00	Descriptor 0 destination end memory start address

8.4.1.7 Descriptor 1 Destination Address Register

Table 156 • HPDMAD1DAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMADAR_DCP1_DST_ADRS	0x00	Descriptor 1 destination end memory start address

8.4.1.8 Descriptor 2 Destination Address Register

Table 157 • HPDMAD2DAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMADAR_DCP2_DST_ADRS	0x00	Descriptor 2 destination end memory start address

8.4.1.9 Descriptor 3 Destination Address Register

Table 158 • HPDMAD3DAR_REG

Bit Number	Name	Reset Value	Description
31:0	HPDMADAR_DCP3_DST_ADRS	0x00	Descriptor 3 destination end memory start address

8.4.1.9.1 Notes on the Destination Address Register (DAR)

- Address is word aligned at the start.
- Address increments on each successful transfer at the destination end.
- HPDMA controller starts reading the data from source memory and transfers to destination memory.
- Software can write all 32-bit destination addresses to prevent non-word aligned transfers at the start and 2 LSBs, 1:0, are masked in the hardware.
- The destination address will be updated in the same field when the descriptor transfer is in progress.

8.4.1.10 Descriptor 0 Control Register

Table 159 • HPDMAD0CR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMACR_DCP0_XFR_SIZE	0	Descriptor 0 transfer size in bytes. Defines number of bytes to be transferred in a descriptor 0 transfer. All zeros in this field indicates 64-KB transfers. As all the transfers are word aligned, 2 LSBs 1:0 are ignored.
16	HPDMACR_DCP_VALID[0]	0	1: Indicates the descriptor 0 is valid and ready to transfer. On completing descriptor 0 transfer, the HPDMA controller clears this bit. Once the descriptor valid bit is set, descriptor fields such as Source address, Destination Address, Transfer size, and Descriptor Valid bits cannot be overwritten. When this bit is set, HPDMA clears the status of the previous transfer, which includes transfer complete, transfer error interrupts, and corresponding descriptor 0 Status Register.
17	HPDMACR_XFR_DIR[0]	0	Descriptor 0 data transfer direction. 0: AHB bus matrix to MSS DDR bridge 1: MSS DDR bridge to AHB bus matrix
18	HPDMACR_DCP_CLR[0]	0	When this bit is set, HPDMA clears the descriptor 0 fields. HPDMA terminates the current transfer and reset descriptor status and control registers. This bit is always read back as zero.
19	HPDMACR_DCP_PAUSE[0]	0	1: HPDMA pauses descriptor 0 transfers, does idle transfers. 0: HPDMA resumes descriptor 0 transfers from where they have stopped.
20	HPDMACR_XFR_CMP_INT[0]	0	1: HPDMA asserts interrupt on completion of descriptor 0 transfers without error. 0: HPDMA will not generate transfer complete interrupt.
21	HPDMACR_XFR_ERR_INT[0]	0	1: HPDMA asserts transfer error interrupt on error during descriptor 0 transfers. 0: HPDMA will not generate transfer error interrupt.
22	HPDMACR_NON_WORD_INT[0]	0	Non-word interrupt enable. 1: HPDMA asserts transfer error interrupt when non-word aligned transfer size is programmed in HPDMACR_DCP0_XFR_SIZE and HPDMA continues the same descriptor transfer. 0: HPDMA will not generate interrupt.
31:23	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.11 Descriptor 1 Control Register

Table 160 • HPDMAD1CR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMACR_DCP1_XFR_SIZE	0	Descriptor 1 transfer size in bytes. Defines number of bytes to be transferred in a descriptor 1 transfer. All zeroes in this field indicates 64-KB transfers. As all the transfers are word aligned, the 2 LSBs 1:0 are ignored.
16	HPDMACR_DCP_VALID[1]	0	1: Indicates the descriptor 1 is valid and ready to transfer. On completing a descriptor 1 transfer, the HPDMA controller clears this bit. Once the descriptor valid bit is set, descriptor fields such as Source Address, Destination Address, Transfer Size, and Descriptor Valid bits cannot be overwritten. When this bit is set, HPDMA clears the status of the previous transfer, which includes transfer complete, transfer error interrupts, and corresponding descriptor 1 Status register.
17	HPDMACR_XFR_DIR[1]	0	Descriptor 2 data transfer direction: 0: AHB bus matrix to MSS DDR bridge 1: MSS DDR bridge to AHB bus matrix
18	HPDMACR_DCP_CLR[1]	0	When this bit is set, HPDMA clears the descriptor 1 fields. HPDMA terminates the current transfer and resets descriptor status and control registers. This bit is always read back as zero.
19	HPDMACR_DCP_PAUSE[1]	0	1: HPDMA pauses Descriptor 1 transfers, does idle transfers. 0: HPDMA resumes descriptor 1 transfers from where they have stopped.
20	HPDMACR_XFR_CMP_INT[1]	0	1: HPDMA asserts interrupt on completion of descriptor 1 transfers without error. 0: HPDMA will not generate transfer complete interrupt.
21	HPDMACR_XFR_ERR_INT[1]	0	1: HPDMA asserts transfer error interrupt on error during descriptor 1 transfers. 0: HPDMA will not generate transfer error interrupt.
22	HPDMACR_NON_WORD_INT[1]	0	Non-word interrupt enable 1: HPDMA asserts transfer error interrupt when non-word aligned transfer size is programmed in HPDMACR_DCP1_XFR_SIZE and HPDMA continues the same descriptor transfer. 0: HPDMA will not generate interrupt.
31:23	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.12 Descriptor 2 Control Register

Table 161 • HPDMAD2CR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMACR_DCP2_XFR_SIZE	0	Descriptor 2 transfer size in bytes. Defines number of bytes to be transferred in a Descriptor 2 transfer. All zeroes in this field indicates 64 KB transfers. As all the transfers are word aligned, 2 LSBs 1:0 are ignored.
16	HPDMACR_DCP_VALID[2]	0	1: Indicates the descriptor 2 is valid and ready to transfer. On completing descriptor 2 transfer, the HPDMA controller clears this bit. Once the descriptor valid bit is set, descriptor fields such as Source Address, Destination Address, Transfer Size, and Descriptor Valid bits cannot be overwritten. When this bit is set, HPDMA clears the status of the previous transfer, which includes transfer complete, transfer error interrupts, and corresponding descriptor 2 Status register.
17	HPDMACR_XFR_DIR[2]	0	Descriptor 2 data transfer direction. 0: AHB bus matrix to MSS DDR bridge 1: DDR bridge to AHB bus matrix
18	HPDMACR_DCP_CLR[2]	0	When this bit is set, HPDMA clears the descriptor 2 fields. HPDMA terminates the current transfer and reset descriptor status and control registers. This bit is always read back as zero.
19	HPDMACR_DCP_PAUSE[2]	0	1: HPDMA pauses the descriptor 2 transfers, does idle transfers. 0: HPDMA resumes descriptor 2 transfers from where they have stopped.
20	HPDMACR_XFR_CMP_INT[2]	0	1: HPDMA asserts interrupt on completion of descriptor 2 transfers without error. 0: HPDMA will not generate transfer complete interrupt.
21	HPDMACR_XFR_ERR_INT[2]	0	1: HPDMA asserts transfer error interrupt on error during descriptor 2 transfers. 0: HPDMA will not generate transfer error interrupt.
22	HPDMACR_NON_WORD_INT[2]	0	Non-word interrupt enable. 1: HPDMA asserts transfer error interrupt when non-word aligned transfer size is programmed in HPDMACR_DCP2_XFR_SIZE and HPDMA continues the same descriptor transfer. 0: HPDMA will not generate interrupt.
31:23	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.13 Descriptor 3 Control Register

Table 162 • HPDMAD3CR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMACR_DCP3_XFR_SIZE	0	Descriptor 3 transfer size in bytes. Defines number of bytes to be transferred in a descriptor 3 transfer. All zeroes in this field indicates 64-KB transfers. As all the transfers are word aligned, 2 LSBs, 1:0, are ignored.
16	HPDMACR_DCP_VALID[3]	0	1: Indicates descriptor 3 is valid and ready to transfer. On completing descriptor 3 transfer, the HPDMA controller clears this bit. Once the descriptor valid bit is set, descriptor fields such as Source address, Destination Address, Transfer size, and Descriptor Valid bits cannot be overwritten. When this bit is set, HPDMA clears the status of the previous transfer, which includes transfer complete, transfer error interrupts, and corresponding descriptor 3 Status register.
17	HPDMACR_XFR_DIR[3]	0	Descriptor 3 data transfer direction. 0: AHB bus matrix to MSS DDR bridge 1: MSS DDR bridge to AHB bus matrix
18	HPDMACR_DCP_CLR[3]	0	When this bit is set, HPDMA clears the descriptor 3 fields. HPDMA terminates the current transfer and resets descriptor status and control registers. This bit is always read back as zero.
19	HPDMACR_DCP_PAUSE[3]	0	1: HPDMA pauses the descriptor 3 transfers, does idle transfers. 0: HPDMA resumes the descriptor 3 transfers from where they have stopped.
20	HPDMACR_XFR_CMP_INT[3]	0	1: HPDMA asserts interrupt on completion of descriptor 3 transfers without error. 0: HPDMA will not generate transfer complete interrupt.
21	HPDMACR_XFR_ERR_INT[3]	0	1: HPDMA asserts transfer error interrupt on error during descriptor 3 transfers. 0: HPDMA will not generate transfer error interrupt.
22	HPDMACR_NON_WORD_INT[3]	0	Non-word interrupt enable. 1: HPDMA asserts transfer error interrupt when non-word aligned transfer size is programmed in HPDMACR_DCP3_XFR_SIZE and HPDMA continues the same descriptor transfer. 0: HPDMA will not generate interrupt
31:23	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.14 Descriptor 0 Status Register

Table 163 • HPDMAD0SR_REG

Bit Number	Name	Reset Value	Description
0	HPDMASR_DCP_ACTIVE[0]	0	Descriptor 0 transfer in progress. 1: Descriptor 0 transfer in progress. 0: Descriptor 0 is in queue when HPDMACR_DCP_VALID[0] bit is set in descriptor 0 Control register.
1	HPDMASR_DCP_CMPLT[0]	0	Descriptor 0 transfer complete. 1: Descriptor 0 transfer completed successfully 0: Descriptor 0 transfer not completed This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[0] of the descriptor 0 control register.
2	HPDMASR_DCP_SERR[0]	0	Descriptor 0 source transfer error. 1: Descriptor 0 transfer error occurred at source end. 0: No error at source end during descriptor 0 transfer This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[0] of the descriptor 0 Interrupt Clear register.
3	HPDMASR_DCP_DERR[0]	0	Descriptor 0 destination transfer error. 1: Descriptor 0 transfer error 0: No error at destination end during descriptor 0 transfer. This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[0] of the descriptor 0 Interrupt Clear register.
31:4	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.15 Descriptor 1 Status Register

Table 164 • HPDMAD1SR_REG

Bit Number	Name	Reset Value	Description
0	HPDMASR_DCP_ACTIVE[1]	0	Descriptor 1 transfer in progress. 1: Descriptor 1 transfer in progress. 0: Descriptor 1 is in queue when HPDMACR_DCP_VALID[1] bit is set in Descriptor 1 Control register.
1	HPDMASR_DCP_CMPLT[1]	0	Descriptor 1 transfer complete. 1: Descriptor 1 transfer completed successfully 0: Descriptor 1 transfer not completed This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[1] of the descriptor 1 Control register.
2	HPDMASR_DCP_SERR[1]	0	Descriptor 1 source transfer error. 1: Descriptor 1 transfer error occurred at source end 0: No error at source end during descriptor 1 transfer This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[1] of the descriptor 1 Interrupt Clear register.

Table 164 • HPDMAD1SR_REG (continued)

Bit Number	Name	Reset Value	Description
3	HPDMASR_DCP_DERR[1]	0	Descriptor 1 destination transfer error. 1: Descriptor 1 transfer error 0: No error at destination end during descriptor 1 transfer This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[1] of the descriptor 1 Interrupt Clear register.
31:4	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.16 Descriptor 2 Status Register

Table 165 • HPDMAD2SR_REG

Bit Number	Name	Reset Value	Description
0	HPDMASR_DCP_ACTIVE[2]	0	Descriptor 2 Transfer in progress. 1: Descriptor 2 transfer in progress. 0: Descriptor 2 is in queue when HPDMACR_DCP_VALID[2] bit is set in descriptor 2 Control register.
1	HPDMASR_DCP_CMPLT[2]	0	Descriptor 2 transfer complete. 1: Descriptor 2 transfer completed successfully. 0: Descriptor 2 transfer not completed. This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[2] of the descriptor 2 Control register.
2	HPDMASR_DCP_SERR[2]	0	Descriptor 2 source transfer error. 1: Descriptor 2 transfer error occurred at source end 0: No error at source end during descriptor 2 transfer This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[2] of the descriptor 2 Interrupt Clear register.
3	HPDMASR_DCP_DERR[2]	0	Descriptor 2 destination transfer error. 1: Descriptor 2 transfer error 0: No error at destination end during descriptor 2 transfer This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[2] of the descriptor 2 Interrupt Clear register.
31:4	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.17 Descriptor 3 Status Register

Table 166 • HPDMAD3SR_REG

Bit Number	Name	Reset Value	Description
0	HPDMASR_DCP_ACTIVE[3]	0	Descriptor 3 transfer in progress. 1: Descriptor 3 transfer in progress. 0: Descriptor 3 is in queue when HPDMACR_DCP_VALID[3] bit is set in descriptor 3 Control register.
1	HPDMASR_DCP_CMPLT[3]	0	Descriptor 2 transfer complete. 1: Descriptor 3 transfer completed successfully. 0: Descriptor 3 transfer not completed. This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[3] of the descriptor 3 Control register.
2	HPDMASR_DCP_SERR[3]	0	Descriptor 3 source transfer error. 1: Descriptor 3 transfer error occurred at source end. 0: No error at source end during descriptor 3 transfer This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[3] of the descriptor 3 Interrupt Clear register.
3	HPDMASR_DCP_DERR[3]	0	Descriptor 3 destination transfer error. 1: Descriptor 3 transfer error 0: No error at destination end during descriptor 3 transfer This bit clears on writing '1' to HPDMAICR_CLR_XFR_INT[3] of the descriptor 3 Interrupt Clear register
31:4	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.18 Descriptor 0 Pending Transfers Register

Table 167 • HPDMAD0PTR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMAPTR_D0_SRC_PNDNG	0	Descriptor 0 source pending transfers in words. This register indicates internal transfer size counter corresponding to source end of the descriptor 0. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the source during the descriptor 0 transfer.
31:16	HPDMAPTR_D0_DST_PNDNG	0	Descriptor 0 destination pending transfers in words. This register indicates the internal transfer size counter corresponding to the destination end of descriptor 0. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the destination during the descriptor 0 transfer.

8.4.1.19 Descriptor 1 Pending Transfers Register

Table 168 • HPDMAD1PTR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMAPTR_D1_SRC_PNDNG	0	Descriptor 1 source pending transfers in words. This register indicates the internal transfer size counter corresponding to the source end of descriptor 1. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the source during descriptor 1 transfer.
31:16	HPDMAPTR_D1_DST_PNDNG	0	Descriptor 1 destination pending transfers in words. This register indicates the internal transfer size counter corresponding to the destination end of descriptor 1. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the destination during descriptor 1 transfer.

8.4.1.20 Descriptor 2 Pending Transfers Register

Table 169 • HPDMAD2PTR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMAPTR_D2_SRC_PNDNG	0	Descriptor 2 source pending transfers in words. This register indicates the internal transfer size counter corresponding to the source end of descriptor 2. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the source during descriptor 2 transfer.
31:16	HPDMAPTR_D2_DST_PNDNG	0	Descriptor 2 destination pending transfers in words. This register indicates the internal transfer size counter corresponding to the destination end of descriptor 2. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the destination during descriptor 2 transfer.

8.4.1.21 Descriptor 3 Pending Transfers Register

Table 170 • HPDMAD3PTR_REG

Bit Number	Name	Reset Value	Description
15:0	HPDMAPTR_D3_SRC_PNDNG	0	Descriptor 3 source pending transfers in words. This register indicates the internal transfer size counter corresponding to the source end of descriptor 3. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the source during descriptor 0 transfer.
31:16	HPDMAPTR_D3_DST_PNDNG	0	Descriptor 3 destination pending transfers in words. This register indicates the internal transfer size counter corresponding to the destination end of descriptor 3. At the end of the transfer, zero in this register indicates the successful transfer, and a non-zero value indicates error occurrence at the destination during descriptor 3 transfer.

8.4.1.22 HPDMA Interrupt Clear Register

Table 171 • HPDMAICR_REG

Bit Number	Name	Reset Value	Description
0	HPDMAICR_CLR_XFR_INT[0]	0	When this bit is set, HPDMA clears the following register bits: Descriptor 0 Status register HPDMASR_DCP_CMPLT[0] HPDMASR_DCP_DERR[0] HPDMASR_DCP_SERR[0] HPDMA Empty Descriptor register HPDMAEDR_DCP_NON_WORD_ERR[0]
1	HPDMAICR_CLR_XFR_INT[1]	0	When this bit is set, HPDMA clears the following register bits: Descriptor 1 Status register HPDMASR_DCP_CMPLT[1] HPDMASR_DCP_DERR[1] HPDMASR_DCP_SERR[1] HPDMA Empty Descriptor register HPDMAEDR_DCP_NON_WORD_ERR[1]
2	HPDMAICR_CLR_XFR_INT[2]	0	When this bit is set, HPDMA clears the following register bits: Descriptor 2 Status register HPDMASR_DCP_CMPLT[2] HPDMASR_DCP_DERR[2] HPDMASR_DCP_SERR[2] HPDMA Empty Descriptor register HPDMAEDR_DCP_NON_WORD_ERR[2]
3	HPDMAICR_CLR_XFR_INT[3]	0	When this bit is set, HPDMA clears the following register bits: Descriptor 3 Status register HPDMASR_DCP_CMPLT[3] HPDMASR_DCP_DERR[3] HPDMASR_DCP_SERR[3] HPDMA Empty Descriptor register HPDMAEDR_DCP_NON_WORD_ERR[3]

Table 171 • HPDMAICR_REG (continued)

Bit Number	Name	Reset Value	Description
4	HPDMAICR_NON_WORD_INT[0]	0	When this bit is set, HPDMA clears the HPDMAEDR_DCP_NON_WORD_ERR[0] bit in the Empty Descriptor register. These bits always read back as 0.
5	HPDMAICR_NON_WORD_INT[1]	0	When this bit is set, HPDMA clears the HPDMAEDR_DCP_NON_WORD_ERR[1] bit in the Empty Descriptor Register. These bits always read back as 0.
6	HPDMAICR_NON_WORD_INT[2]	0	When this bit is set, HPDMA clears the HPDMAEDR_DCP_NON_WORD_ERR[2] bit in the Empty Descriptor register. These bits always read back as 0.
7	HPDMAICR_NON_WORD_INT[3]	0	When this bit is set, HPDMA clears the HPDMAEDR_DCP_NON_WORD_ERR[3] bit in the Empty Descriptor register. These bits always read back as 0.
31:8	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.4.1.23 HPDMA Debug Register

Table 172 • HPDMADR_REG

Bit Number	Name	Reset Value	Description
0	HPDMADR_BFR_EMPTY	1	Data buffer is empty; HPDMA controller initiates idle transfers on the destination memory end. 1: Data buffer is empty. 0: Data Buffer is not empty.
1	HPDMADR_BFR_FULL	0	Data buffer is full; HPDMA controller initiates idle transfers on the source memory end. 1: Data buffer is full. 0: Data buffer is not full.
4:2	HPDMADR_BFR_RD_PNTR[2:0]	0	HPDMA data buffer read pointer
7:5	HPDMADR_BFR_WR_PNTR[2:0]	0	HPDMA data buffer write pointer
11:8	HPDMADR_AHM1_CST_DBG[3:0]	0	Master 1 (AHB bus matrix) current state 0001 – IDLE 0010 – WRITE 0100 – READ 1000 – WAIT
15:12	HPDMADR_AHM2_CST_DBG[3:0]	0	Master 2 (MSS DDR bridge) current state 0001 – IDLE 0010 – WRITE 0100 – READ 1000 – WAIT

Table 172 • HPDMADR_REG (continued)

Bit Number	Name	Reset Value	Description
18:16	HPDMADR_WBC_CST_DBG[2:0]	0	Write buffer controller current state 001 – IDLE 010 – RUN 100 – WAIT
21:19	HPDMADR_RBC_CST_DBG[2:0]	0	Read buffer controller current state 001 – IDLE 010 – RUN 100 – WAIT
25:22	HPDMADR_RRBN_CST_DBG[3:0]	0	Round robin FSM current state 0001 – D0 0010 – D1 0100 – D2 1000 – D3
27:26	HPDMADR_DMA_CST_DBG[1:0]	0	DMA controller FSM current state 01 – IDLE 10 – RUN
31:28	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

8.5 SYSREG Control Register

In addition to the specific HPDMA registers, the registers provided in [Table 173](#), page 263 also control the behavior of the HPDMA peripheral. Refer to the [System Register Block](#), page 670 for a detailed description of each register and associated bits.

Table 173 • SYSREG Control Registers

Register Name	Register Type	Flash Write Protect	Reset Source	Description
SOFT_RESET_CR	RW-P	Bit	SYSRESET_N	Bit 17 is used for HPDMA reset '1' – Reset HPDMA '0' – Release from HPDMA reset
MASTER_WEIGHT1_CR	RW-P	Register	SYSRESET_N	Bits 4:0 define round robin weight values for the HPDMA master.

9 Peripheral DMA

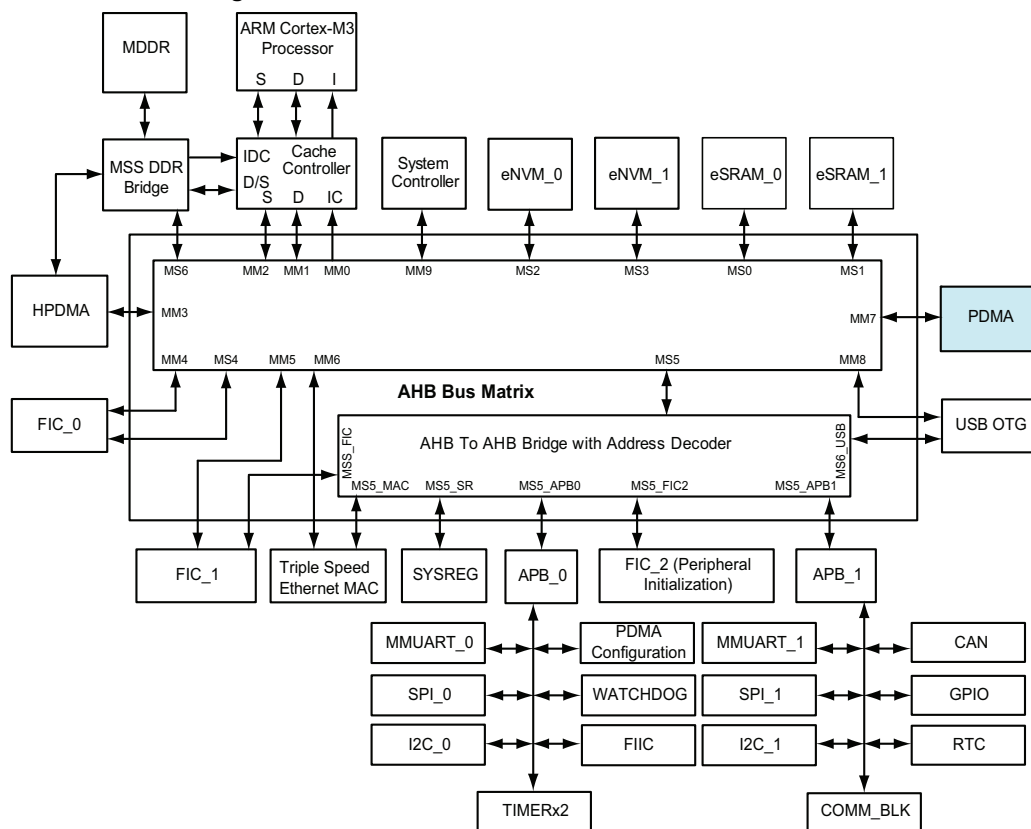
The peripheral direct memory access (PDMA) is an AHB master associated with the AHB bus matrix, as shown in the following figure. The PDMA allows data transfers from various MSS peripherals to memory, memory to various peripherals, and memory to memory. The peripheral can be MSS peripheral (MMUART, CAN, SPI, and COMM_BLK) or fabric peripheral (FIC_0 and FIC_1). The memory can be MSS memory (eNVM_0, eNVM_1, eSRAM_0, and eSRAM_1) or fabric internal memory (LSRAM and uSRAM) or external memory connected to fabric.

The PDMA is typically used for off loading byte-intensive operations and involving peripherals from the Cortex-M3 processor. For example, the firmware could direct the PDMA to transfer the next 1,000 characters received on one of the MMUARTs to eSRAM and notify the processor when ready.

9.1 Features

- Up to 8 DMA channels
- Ping-pong mode support
- Channels priority designations
- Memory to memory DMA capable
- Interrupt capability

Figure 129 • PDMA Interfacing with AHB Bus Matrix



9.2 Functional Description

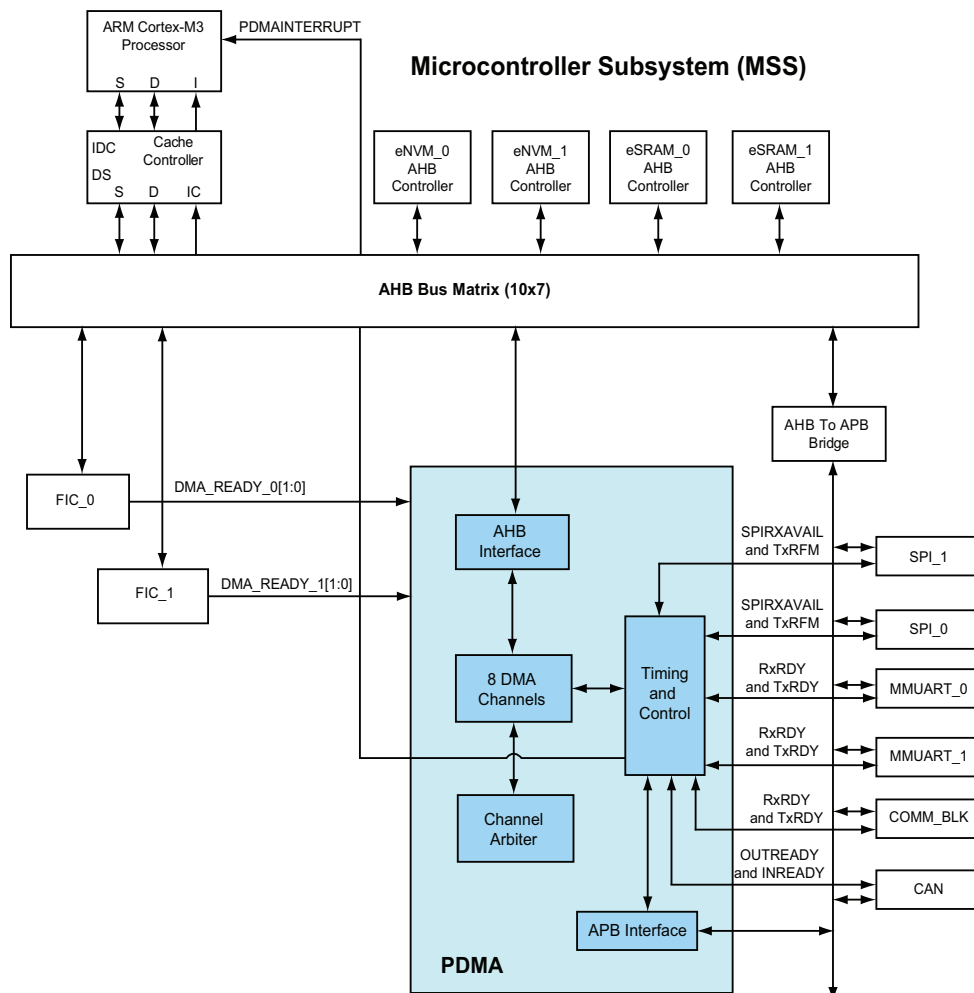
This section provides the detailed description of the PDMA.

9.2.1 Architecture Overview

The PDMA controller mainly consists of the following blocks, as shown in the following figure.

- AHB and APB Interfaces
- 8-Channel DMA Controller
- Timing and Control
- Channel Arbiter

Figure 130 • PDMA Internal Architecture



9.2.1.1 AHB and APB Interfaces

As shown in the preceding figure, the PDMA has two interfaces—AHB-lite and APB. The APB interface is a 32-bit APB slave used for configuring the PDMA. DMA operations do not occur on the APB bus interface of the PDMA. The PDMA performs single cycle accesses on the AHB interface only and Burst mode is not supported.

9.2.1.2 8-Channel DMA Controller

The 8-channel DMA controller consists of eight instances of a single DMA channel, as shown in the preceding figure. Each channel can be configured to perform 8-bit, 16-bit, or 32-bit data transfers from the peripheral to memory, memory to peripheral, and memory to memory. Each DMA channel supports Ping-pong mode for continuous data transfer. To enable and use PDMA services, the AHB bus master matrix must configure the following 32-bit registers.

- CHANNEL_x_CONTROL
- CHANNEL_x_BUFFER_A_SRC_ADDR
- CHANNEL_x_BUFFER_A_DST_ADDR
- CHANNEL_x_BUFFER_A_TRANSFER_COUNT
- CHANNEL_x_BUFFER_B_SRC_ADDR
- CHANNEL_x_BUFFER_B_DST_ADDR
- CHANNEL_x_BUFFER_B_TRANSFER_COUNT

If bidirectional DMA of peripheral to memory (receive) and memory to peripheral (transmit) is desired, two channels must be programmed appropriately. In particular, the TRANSFER_SIZE fields in both the CHANNEL_x_CONTROL registers must be programmed identically. The PDMA performs the correct byte lane adjustments appropriate to the address being used on the AHB. Efficient use of memory storage is achieved in this manner, even if only performing byte or 16-bit accesses to or from a peripheral. For example, when the PDMA is accessing peripherals, the lowest 8 or 16 bits of the data bus are always used for 8-bit or 16-bit transfers. For 32-bit transfers, the full 32-bits are used. It is possible to configure the data width of a transfer to be independent of the address increment. The address increment at both ends of the DMA transfer can be different, which is required when reading from a peripheral holding register (single address) and writing to memory incrementally (many addresses).

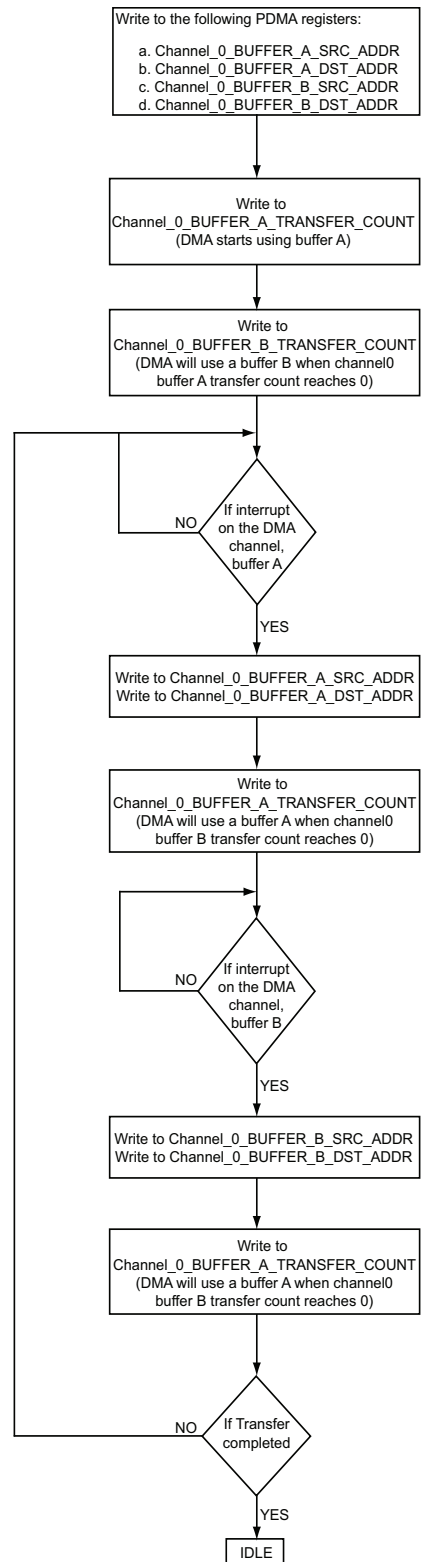
Sixteen possible channels are available to the PDMA, as listed below. Only eight are used simultaneously.

- MMUART_0 to any MSS memory-mapped location
- Any MSS memory-mapped location to MMUART_0
- MMUART_1 to any MSS memory-mapped location
- Any MSS memory-mapped location to MMUART_1
- SPI_0 to any MSS memory-mapped location
- Any MSS memory-mapped location to SPI_0
- SPI_1 to any MSS memory-mapped location
- Any MSS memory-mapped location to SPI_1
- FPGA fabric peripheral on FIC_0 to any MSS memory-mapped location
- Any MSS memory-mapped location to FPGA fabric peripheral on FIC_0
- CAN to any MSS memory-mapped location
- Any MSS memory-mapped location to CAN
- FPGA fabric peripheral on FIC_1 to any MSS memory-mapped location
- Any MSS memory-mapped location to FPGA fabric peripheral on FIC_1
- COMM_BLK to any MSS memory-mapped location
- Any MSS memory-mapped locations to COMM_BLK

9.2.1.2.1 Ping-Pong Mode

Ping-pong mode is a dual buffering scheme for continuous stream of operation. There are two buffers (Buffer A and Buffer B) associated with each DMA channel for ping-pong operation. This removes the real-time constraint on the firmware of having to service the DMA channel in real time, which would exist if there were only one DMA buffer per channel.

To begin a transaction, source address, destination address, and transfer size in bytes of buffer A and buffer B are to be configured by the AHB bus matrix master (such as Cortex-M3 firmware). The following figure shows the sequence of operations that must be performed by firmware for ping-pong operation on a configured DMA channel. The channel control register (CHANNEL_x_CONTROL) is configured initially before enabling ping-pong operation.

Figure 131 • Flow of Ping-Pong Operation on DMA Channel

9.2.1.3 Timing and Control

The peripheral ready signals (for example, RxRDY and TxRDY in MMUART) from the SPI, MMUART, COMM_BLK, and CAN are directly connected to the PDMA. The ready signals from the CAN are not used and are tied to logic 1 internally. The PDMA takes care of writing or reading the receive or transmit holding registers within each peripheral using the APB interface.

The DMAREADY_0 and DMAREADY_1 signals correspond to the ready signals from the fabric peripheral. If the channel is configured for peripheral DMA and the direction is from the fabric peripheral to memory, this signal indicates that the fabric peripheral can write data to memory. If the channel is configured for peripheral DMA and the direction is from memory to the fabric peripheral, this signal indicates that the fabric peripheral can read data from memory. The PDMA does not support peripheral to peripheral data transfer, scatter-gather DMA, and I2C DMA.

9.2.1.4 Channel Arbiter

The channel arbiter is an arbitration algorithm used to service the channels based on the priority, as shown in [Figure 130](#), page 265. By default, all channels have equal priority. To configure the PDMA channel priority, RATIO_HIGH_LOW register must be configured by the AHB bus matrix master. The RATIOHILO field in the RATIO_HIGH_LOW register indicates the ratio of high priority requests to low priority requests. For example, a RATIOHILO value of 3:1 means that a high priority DMA channel has 3 DMA access opportunities for every one access of a low priority DMA channel.

When the RATIOHILO value is set to 0, both high and low priority requests are serviced in a round robin fashion.

The following table lists the valid values for RATIOHILO. All other values are reserved.

Table 174 • RATIOHILO Field Definition

Value	High:Low Ratio	Comments
0		Round robin
1	1:1	Ping-pong between high and low priority requests
3	3:1	3 high to 1 low
7	7:1	7 high to 1 low
15	15:1	15 high to 1 low
31	31:1	31 high to 1 low
63	63:1	63 high to 1 low
127	127:1	127 high to 1 low
255	255:1	255 high to 1 low
All others		Reserved

Refer to [PDMA Register Map](#), page 275 for more information on configuring the register.

9.2.2 Port List

Table 175 • Port List

Name	Type	Polarity	Description
DMAREADY_FIC_0	Input	High	
DMAREADY_FIC_0	Input	High	

9.2.3 Initialization

To initiate and setup DMA transactions, PDMA has to be initialized. The initialization process starts with a reset sequence followed by Channel configuration and interrupt configuration.

9.2.3.1 Reset

The PDMA registers are reset on power-up. The PDMA can be reset by configuring the following

- Bit 5 of `SOFT_RESET_CR` system register.
- Bit 5 of the `CHANNEL_x_CONTROL` register for channel reset.

9.2.3.2 Channel Configuration

Before configuring each PDMA channel, the round robin weight is specified if needed, using the `MASTER_WEIGHT_CR` register or configuring the AHB bus matrix in Libero SoC.

To configure each PDMA channel, following fields of the channel control register has to be set:

- Peripheral select - bits[26:23] of `CHANNEL_x_CONTROL`
- No. of wait states - bits[21:14] of `CHANNEL_x_CONTROL`
- Source and/or destination address increment - bits[13:10] of `CHANNEL_x_CONTROL`
- Channel priority - bit 9 of `CHANNEL_x_CONTROL`
- Interrupt enable - bit 6 of `CHANNEL_x_CONTROL`
- Transfer size - bits [3:2] of `CHANNEL_x_CONTROL`
- Direction - bit 1 of `CHANNEL_x_CONTROL`
- Select the data Transfer type - bit 0 of `CHANNEL_x_CONTROL`

9.2.3.3 Interrupt

To use PDMA interrupt to Cortex-M3, Bit 8 of `INTERRUPT_ENABLE0` register (located at address 0x40006000) has to be set. The PDMA Interrupt signal is also mapped to the dedicated interrupt signal `MSS_INT_M2F[8]` of the fabric interface interrupt controller (FIIC). This is to interrupt the user logic instantiated in the FPGA.

To determine transfer complete interrupt for each channel, the `BUFFER_STATUS_x` register bits[1:0] has to be monitored. The bit 7 and bit 8 of `CHANNEL_x_CONTROL` register are used to clear the transfer complete interrupts of the channel.

9.2.4 Details of Operations

After initialization, the PDMA is ready to function in any one of following transfer modes:

- Peripheral to Memory Transfers/Memory to Memory Transfers
- Posted APB Writes

9.2.4.1 Peripheral to Memory Transfers/Memory to Memory Transfers

For peripheral to memory or peripheral to memory transfer, the DMA transfer starts if `BUFFER_A_TRANSFER_COUNT` or `BUFFER_B_TRANSFER_COUNT` is non-zero.

Before the transfer the source address (`CHANNEL_x_BUFFER_A_SRC_ADDR`) and destination address (`CHANNEL_x_BUFFER_B_DST_ADDR`) of a channel are configured; then write to one of the transfer count registers to begin the DMA transaction. Alternatively, firmware can also write to the control register first and turn pause on, if needed, then turn it off later.

If the `PAUSE` bit in the `CHANNEL_x_CONTROL` register is set, when you write a non-zero value to `BUFFER_A_TRANSFER_COUNT` or `BUFFER_B_TRANSFER_COUNT`, then the DMA transaction waits until `PAUSE` is cleared.

If bidirectional DMA of peripheral to memory (receive) and memory to peripheral (transmit) is desired, two channels must be programmed appropriately. In particular, the `TRANSFER_SIZE` fields in both the `CHANNEL_x_CONTROL` registers must be programmed identically.

Channels can be assigned to peripherals or memory arbitrarily. For example, to receive only DMA data from one of the SPI ports, only one channel is required. In this case, the `DIR` bit in the

CHANNEL_x_CONTROL register should be set to 0 (peripheral to memory) and the PERIPHERAL_SEL field should be set to 4 (SPI_0 receive to memory).

9.2.4.2 Posted APB Writes

The AHB to APB bridges in the SmartFusion2 device implement posted writes (also known as dump and run) for write accesses to peripherals. PDMA performs a write operation to a peripheral but the data is not written to the peripheral immediately. Therefore, the PDMA block should not start another DMA on this channel based on the state of the ready signal from that peripheral until the write is complete. The time window involved is variable, depending on the ratio of M3_CLK to PCLK, for each of the two peripheral buses. WRITE_ADJ in the CHANNEL_x_CONTROL register is an 8-bit binary coded field used to define, for each DMA channel, how long to wait (in M3_CLKs) after each DMA transfer cycle before interpreting the ready signal for that DMA channel as representing a new request.

A suitable value for WRITE_ADJ depends on the target of the DMA transfer.

The following steps describe how to select the values:

1. The WRITE_ADJ value of 10 can be provided as a default value
 - When the PDMA channel is configured for transfers with MSS peripherals.
 - For DMA transfers with FPGA fabric implemented peripherals, making use of the DMAREADY0 or DMAREADY1 fabric interface signals indicate that the peripheral is ready for another DMA transfer.
2. The WRITE_ADJ parameter can be set to zero to achieve the maximum transfer speed for memory to memory transfers.
3. The internal latency of FPGA implemented peripherals decide the WRITE_ADJ value for fabric peripherals that do not use the DMAREADY0 or DMAREADY1 fabric interface signals.

9.3 How to Use the PDMA

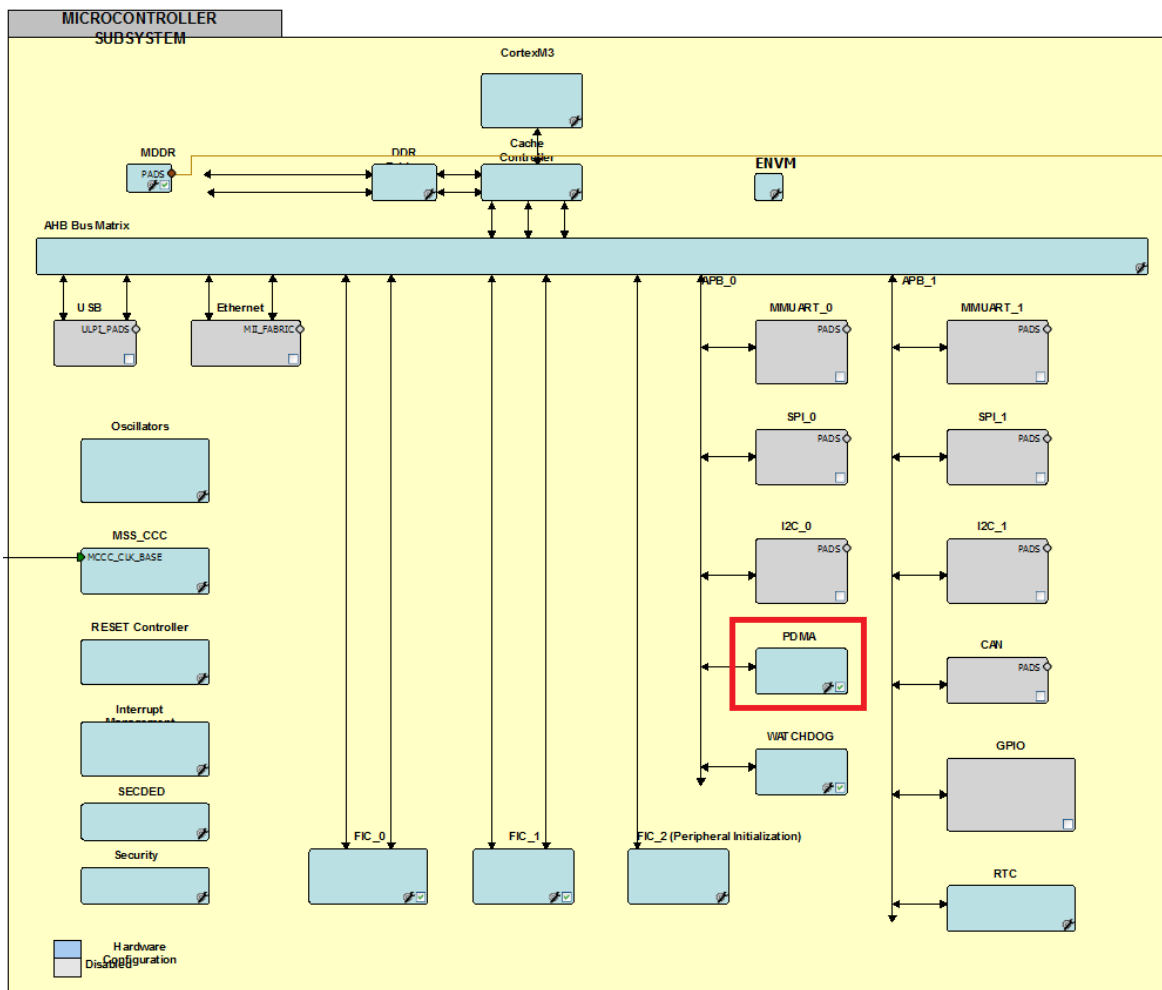
This following sections describe how to use the PDMA in an application.

9.3.1 Design Flow

The following steps are used to enable the PDMA in the application:

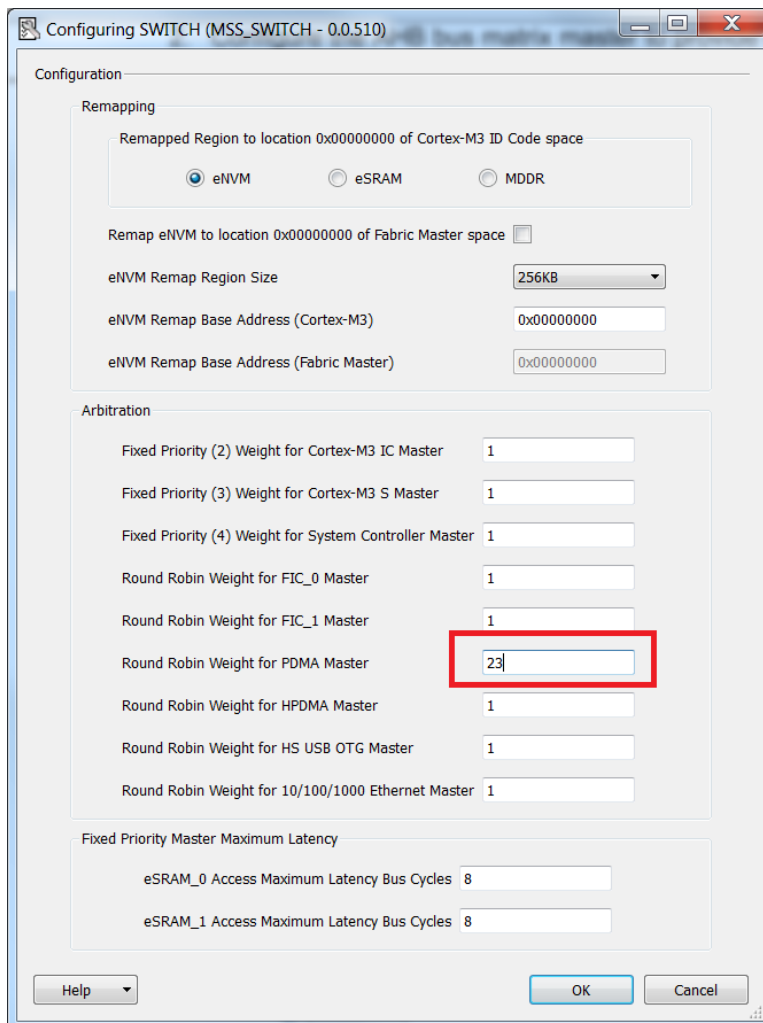
1. Enable PDMA by using the MSS configurator in the application, as shown in the following figure.

Figure 132 • Enable PDMA



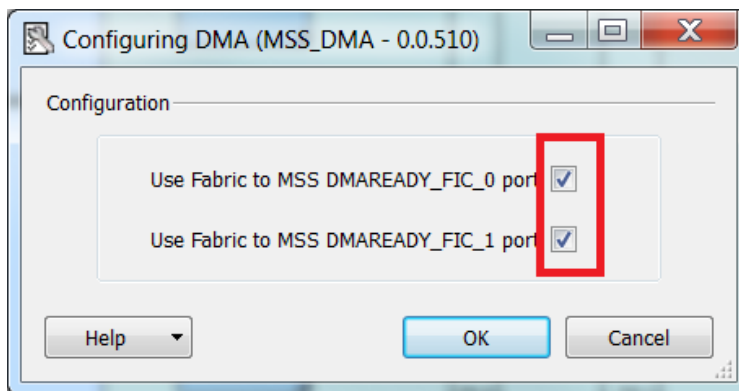
2. Configure the AHB bus matrix master to provide weights to PDMA, as shown in the following figure.

Figure 133 • PDMA AHB Bus Master Matrix Configuration



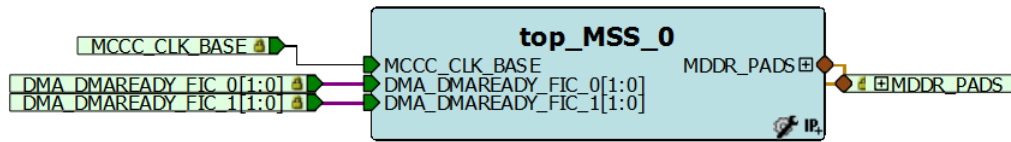
3. To configure the PDMA to transfer data between fabric peripherals (associated on FIC_0 and FIC_1) and MSS memories, select the PDMA configurator, as shown in the following figure.

Figure 134 • PDMA Transfers Data Between FIC and MSS Memory



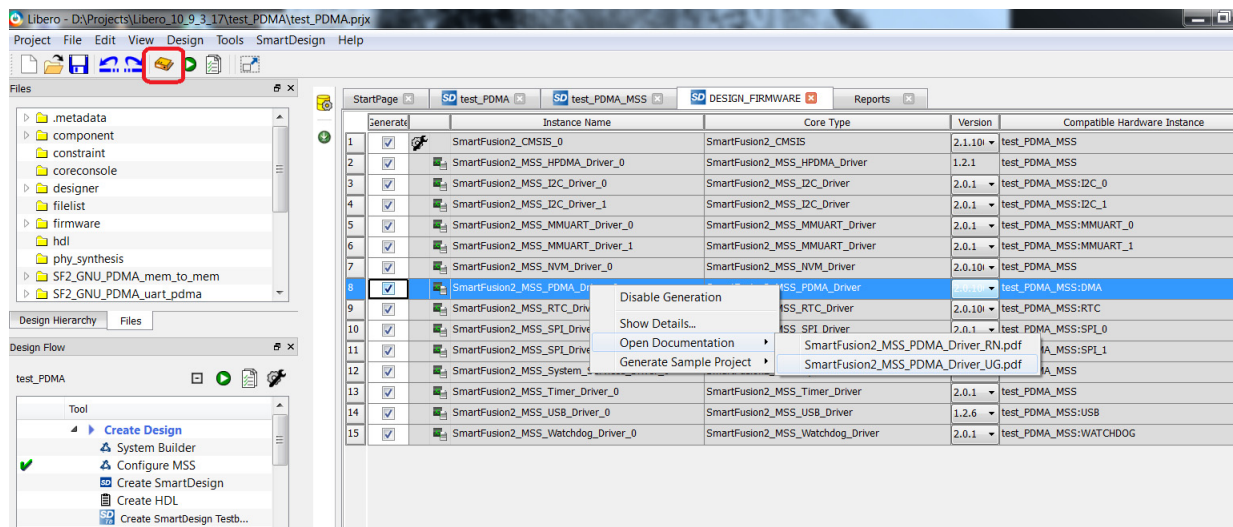
4. The PDMA signals in top level instance are shown in the following figure.

Figure 135 • PDMA Signals



5. Generate the component by clicking **Generate Component** or by selecting **SmartDesign>Generate Component**. For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace is included into the project. Click the highlighted **Configure firmware** button as shown in the following figure to find the PDMA drivers.

Figure 136 • PDMA Driver User Guide



6. Click **Generate Bitstream** under **Program Design** to complete *.fdb file generation. Double-click **Export Firmware** under **Handoff Design for Firmware Development** in the Libero SoC design flow window to generate the SoftConsole Firmware Project. The SoftConsole folder contains the mss_pdma firmware driver. The firmware driver, mss_pdma (mss_pdma.c and mss_pdma.h) which provides a set of functions for controlling the MSS PDMA transfers, can also be downloaded from the Microsemi firmware catalog. the following table lists the APIs for PDMA. For more information on the APIs, refer to the **Smartfusion2_MSS_PDMA_Driver_UG** as shown in the preceding figure.

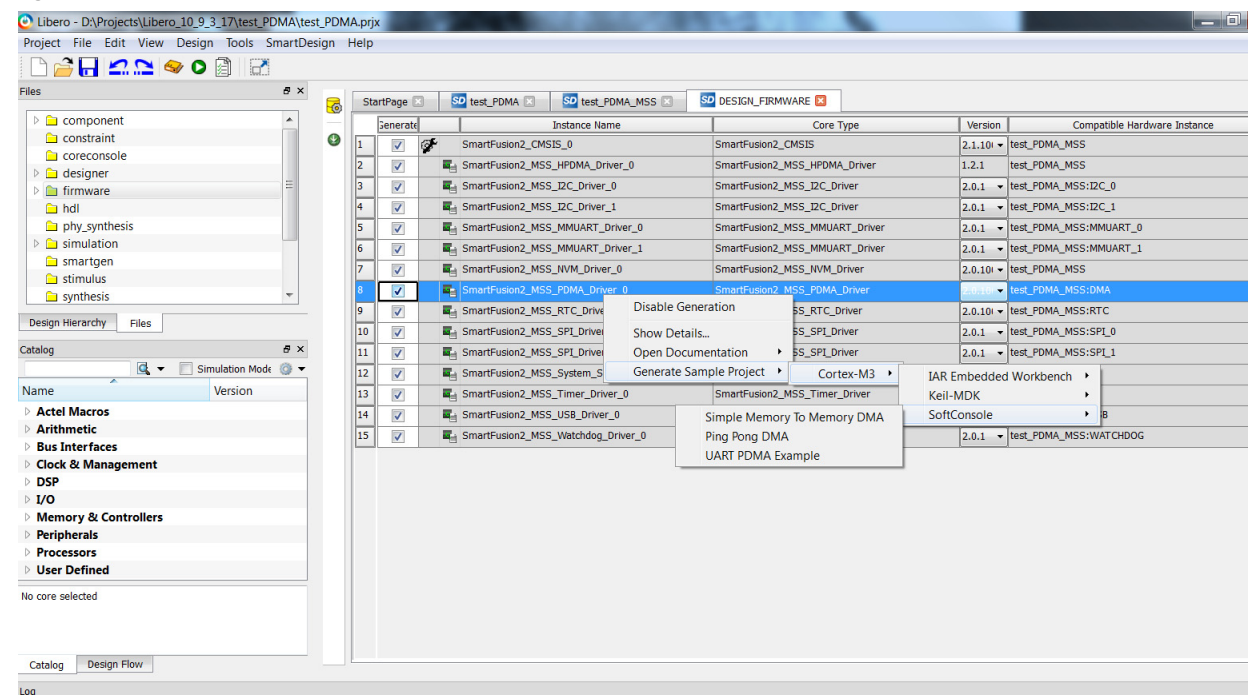
Table 176 • MSS PDMA APIs

Category	API	Description and Usage
Initialization	PDMA_init()	Initializes PDMA
Configuration	PDMA_configure()	Channel priority Transfer size Source and/or destination address increment Source or destination of DMA transfer Insert wait states Selects the peripheral and direction Selects the transfer type
	PDMA_pause()	Pauses PDMA transfer
	PDMA_resume()	Resumes PDMA transfer
	PDMA_abort()	Aborts the PDMA transfer

Table 176 • MSS PDMA APIs (continued)

DMA transfer and control	PDMA_start()	Starts PDMA transfer
	PDMA_load_next_buffer()	Loads with next buffer of data
	PDMA_status()	Gets the status of PDMA transfer
Interrupt control functions	PDMA_set_irq_handler()	Register PDMA channel interrupt handler functions with the driver
	PDMA_enable_irq()	Enables interrupt for Cortex-M3 processor and channel interrupt.
	PDMA_clear_irq()	Clears the Cortex-M3 processor and channel interrupts.
	PDMA_disables_irq()	Disables the interrupts for specified channel

7. For more information on PDMA usage, the sample projects are available and can be generated, as shown in the following figure.

Figure 137 • PDMA Examples

9.3.2 PDMA Use Models

9.3.2.1 Use Model 1: Peripheral to Memory or Memory to Memory

1. Enable PDMA using the MSS configurator of Libero SoC.
2. Initialize the PDMA using PDMA_init().
3. Check the status of PDMA channel using PDMA_status().
4. Clear if any interrupts are pending PDMA_clear_irq().
5. Configure the following using PDMA_configure().
 - Transfer type
 - Selection of the peripheral and direction (if peripheral to memory transfer is selected)
 - Transfer size (data width)
 - Source and destination address increment
 - Channel priority
 - Wait states (WRITE_ADJ)

6. Start the transfer using PDMA_start()
 - Buffer A and buffer B source and destination addresses
 - Transfer count in bytes
7. Enable channel interrupt and Cortex-M3 interrupt using PDMA_enable_irq().
8. Disable the interrupt for channel PDMA_disable_irq().
9. To pause the data transfers, use PDMA_pause().
10. To resume the data transfer, use PDMA_resume().
11. Check for completion of the data transfer using PDMA_status().

Note: The MSS PDMA does not support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for more information.

9.4 PDMA Register Map

The following table summarizes each of the registers covered by this document. The base address is 0x40003000.

Table 177 • SmartFusion2 SoC FPGA PDMA Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
Ratio_HIGH_LOW	0x00	R/W	0	Ratio of high priority transfers versus low priority transfers.
BUFFER_STATUS	0x04	R	0	Indicates when buffers have drained.
CHANNEL_x_CONTROL (X=0)	0x20	R/W	0	Channel 0 Control register
CHANNEL_x_STATUS (X=0)	0x24	R	0	Channel 0 Status register
CHANNEL_x_BUFFER_A_SRC_ADDR (x=0)	0x28	R/W	0	Channel 0 buffer A source address
CHANNEL_x_BUFFER_A_DST_ADDR (x=0)	0x2C	R/W	0	Channel 0 buffer A destination address
CHANNEL_x_BUFFER_A_TRANSFER_COUNT (x=0)	0x30	R/W	0	Channel 0 buffer A transfer count
CHANNEL_x_BUFFER_B_SRC_ADDR (x=0)	0x34	R/W	0	Channel 0 buffer B source address
CHANNEL_x_BUFFER_B_DST_ADDR (x=0)	0x38	R/W	0	Channel 0 buffer B destination address
CHANNEL_x_BUFFER_B_TRANSFER_COUNT (x=0)	0x3C	R/W	0	Channel 0 buffer B transfer count
CHANNEL_1_CONTROL	0x40	R/W	0	Channel 1 Control register
CHANNEL_1_STATUS	0x44	R	0	Channel 1 Status register
CHANNEL_1_BUFFER_A_SRC_ADDR	0x48	R/W	0	Channel 1 buffer A source address
CHANNEL_1_BUFFER_A_DST_ADDR	0x4C	R/W	0	Channel 1 buffer A destination address
CHANNEL_1_BUFFER_A_TRANSFER_COUNT	0x50	R/W	0	Channel 1 buffer A transfer count
CHANNEL_1_BUFFER_B_SRC_ADDR	0x54	R/W	0	Channel 1 buffer B source address
CHANNEL_1_BUFFER_B_DST_ADDR	0x58	R/W	0	Channel 1 buffer B destination address

Table 177 • SmartFusion2 SoC FPGA PDMA Register Map (continued)

Register Name	Address Offset	Register Type	Reset Value	Description
CHANNEL_1_BUFFER_B_TRANSFER_COUNT	0x5C	R/W	0	Channel 1 buffer B transfer count
CHANNEL_2_CONTROL	0x60	R/W	0	Channel 2 Control register
CHANNEL_2_STATUS	0x64	R	0	Channel 2 Status register
CHANNEL_2_BUFFER_A_SRC_ADDR	0x68	R/W	0	Channel 2 buffer A source address
CHANNEL_2_BUFFER_A_DST_ADDR	0x6C	R/W	0	Channel 2 buffer A destination address
CHANNEL_2_BUFFER_A_TRANSFER_COUNT	0x70	R/W	0	Channel 2 buffer A transfer count
CHANNEL_2_BUFFER_B_SRC_ADDR	0x74	R/W	0	Channel 2 buffer B source address
CHANNEL_2_BUFFER_B_DST_ADDR	0x78	R/W	0	Channel 2 buffer B destination address
CHANNEL_2_BUFFER_B_TRANSFER_COUNT	0x7C	R/W	0	Channel 2 buffer B transfer count
CHANNEL_3_CONTROL	0x80	R/W	0	Channel 3 Control register
CHANNEL_3_STATUS	0x84	R	0	Channel 3 Status register
CHANNEL_3_BUFFER_A_SRC_ADDR	0x88	R/W	0	Channel 3 buffer A source address
CHANNEL_3_BUFFER_A_DST_ADDR	0x8C	R/W	0	Channel 3 buffer A destination address
CHANNEL_3_BUFFER_A_TRANSFER_COUNT	0x90	R/W	0	Channel 3 buffer A transfer count
CHANNEL_3_BUFFER_B_SRC_ADDR	0x94	R/W	0	Channel 3 buffer B source address
CHANNEL_3_BUFFER_B_DST_ADDR	0x98	R/W	0	Channel 3 buffer B destination address
CHANNEL_3_BUFFER_B_TRANSFER_COUNT	0x9C	R/W	0	Channel 3 buffer B transfer count
CHANNEL_4_CONTROL	0xA0	R/W	0	Channel 4 Control register
CHANNEL_4_STATUS	0xA4	R	0	Channel 4 Status register
CHANNEL_4_BUFFER_A_SRC_ADDR	0xA8	R/W	0	Channel 4 buffer A source address
CHANNEL_4_BUFFER_A_DST_ADDR	0xAC	R/W	0	Channel 4 buffer A destination address
CHANNEL_4_BUFFER_A_TRANSFER_COUNT	0xB0	R/W	0	Channel 4 buffer A transfer count
CHANNEL_4_BUFFER_B_SRC_ADDR	0xB4	R/W	0	Channel 4 buffer B source address
CHANNEL_4_BUFFER_B_DST_ADDR	0xB8	R/W	0	Channel 4 buffer B destination address
CHANNEL_4_BUFFER_B_TRANSFER_COUNT	0xBC	R/W	0	Channel 4 buffer B transfer count

Table 177 • SmartFusion2 SoC FPGA PDMA Register Map (continued)

Register Name	Address Offset	Register Type	Reset Value	Description
CHANNEL_5_CONTROL	0xC0	R/W	0	Channel 5 Control register
CHANNEL_5_STATUS	0xC4	R	0	Channel 5 Status register
CHANNEL_5_BUFFER_A_SRC_ADDR	0xC8	R/W	0	Channel 5 buffer A source address
CHANNEL_5_BUFFER_A_DST_ADDR	0xCC	R/W	0	Channel 5 buffer A destination address
CHANNEL_5_BUFFER_A_TRANSFER_COUNT	0xD0	R/W	0	Channel 5 buffer A transfer count
CHANNEL_5_BUFFER_B_SRC_ADDR	0xD4	R/W	0	Channel 5 buffer B source address
CHANNEL_5_BUFFER_B_DST_ADDR	0xD8	R/W	0	Channel 5 buffer B destination address
CHANNEL_5_BUFFER_B_TRANSFER_COUNT	0xDC	R/W	0	Channel 5 buffer B transfer count
CHANNEL_6_CONTROL	0xE0	R/W	0	Channel 6 Control register
CHANNEL_6_STATUS	0xE4	R	0	Channel 6 Status register
CHANNEL_6_BUFFER_A_SRC_ADDR	0xE8	R/W	0	Channel 6 buffer A source address
CHANNEL_6_BUFFER_A_DST_ADDR	0xEC	R/W	0	Channel 6 buffer A destination address
CHANNEL_6_BUFFER_A_TRANSFER_COUNT	0xF0	R/W	0	Channel 6 buffer A transfer count
CHANNEL_6_BUFFER_B_SRC_ADDR	0xF4	R/W	0	Channel 6 buffer B source address
CHANNEL_6_BUFFER_B_DST_ADDR	0xF8	R/W	0	Channel 6 buffer B destination address
CHANNEL_6_BUFFER_B_TRANSFER_COUNT	0xFC	R/W	0	Channel 6 buffer B transfer count
CHANNEL_7_CONTROL	0x100	R/W	0	Channel 7 Control register
CHANNEL_7_STATUS	0x104	R	0	Channel 7 Status register
CHANNEL_7_BUFFER_A_SRC_ADDR	0x108	R/W	0	Channel 7 buffer A source address
CHANNEL_7_BUFFER_A_DST_ADDR	0x10C	R/W	0	Channel 7 buffer A destination address
CHANNEL_7_BUFFER_A_TRANSFER_COUNT	0x110	R/W	0	Channel 7 buffer A transfer count
CHANNEL_7_BUFFER_B_SRC_ADDR	0x114	R/W	0	Channel 7 buffer B source address
CHANNEL_7_BUFFER_B_DST_ADDR	0x118	R/W	0	Channel 7 buffer B destination address
CHANNEL_7_BUFFER_B_TRANSFER_COUNT	0x11C	R/W	0	Channel 7 buffer B transfer count

9.4.1 PDMA Configuration Register Bit Definitions

The following registers are present in the PDMA engine:

9.4.1.1 **RATIO_HIGH_LOW** Register Bit Definition

Table 178 • Ratio_HIGH_LOW

Bit Number	Name	Reset Value	Description
[31:8]	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
[0:7]	RATIOHILO	0	This field indicates the ratio of high priority to low priority for DMA access opportunities. This register gives the number of DMA opportunities provided by the channel arbiter to high priority channels for every one opportunity provided to a low priority channel. Only certain values are allowed, as shown in Table 174 , page 268.

9.4.1.2 **BUFFER_STATUS** Register Bit Definition

Table 179 • BUFFER_STATUS

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
15	CH7BUFB	0	If CH_COMP_B for channel 7 is set and if BUF_B_SEL for channel 7 is clear, this bit is asserted.
14	CH7BUFA	0	If CH_COMP_A for channel 7 is set and if BUF_A_SEL for channel 7 is clear, this bit is asserted.
13	CH6BUFB	0	If CH_COMP_B for channel 6 is set and if BUF_B_SEL for channel 6 is clear, this bit is asserted.
12	CH6BUFA	0	If CH_COMP_A for channel 6 is set and if BUF_A_SEL for channel 6 is clear, this bit is asserted.
11	CH5BUFB	0	If CH_COMP_B for channel 5 is set and if BUF_B_SEL for channel 5 is clear, this bit is asserted.
10	CH5BUFA	0	If CH_COMP_A for channel 5 is set and if BUF_A_SEL for channel 5 is clear, this bit is asserted.
9	CH4BUFB	0	If CH_COMP_B for channel 4 is set and if BUF_B_SEL for channel 4 is clear, this bit is asserted.
8	CH4BUFA	0	If CH_COMP_A for channel 4 is set and if BUF_A_SEL for channel 4 is clear, this bit is asserted.
7	CH3BUFB	0	If CH_COMP_B for channel 3 is set and if BUF_B_SEL for channel 3 is clear, this bit is asserted.
6	CH3BUFA	0	If CH_COMP_A for channel 3 is set and if BUF_A_SEL for channel 3 is clear, this bit is asserted.
5	CH2BUFB	0	If CH_COMP_B for channel 2 is set and if BUF_B_SEL for channel 2 is clear, this bit is asserted.
4	CH2BUFA	0	If CH_COMP_A for channel 2 is set and if BUF_A_SEL for channel 2 is clear, this bit is asserted.

Table 179 • BUFFER_STATUS (continued)

Bit Number	Name	Reset Value	Description
3	CH1BUFB	0	If CH_COMP_B for channel 1 is set and if BUF_B_SEL for channel 1 is clear, this bit is asserted.
2	CH1BUFA	0	If CH_COMP_A for channel 1 is set and if BUF_A_SEL for channel 1 is clear, this bit is asserted.
1	CH0BUFB	0	If CH_COMP_B for channel 0 is set and if BUF_B_SEL for channel 0 is clear, this bit is asserted.
0	CH0BUFA	0	If CH_COMP_A for channel 0 is set and if BUF_A_SEL for channel 0 is clear, this bit is asserted.

9.4.1.3 CHANNEL_x_CONTROL Register Bit Definition

Table 180 • CHANNEL_x_CONTROL

Bit Number	Name	Reset Value	Description
[31:27]	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
[26:23]	PERIPHERAL_SEL	0	Selects the peripheral assigned to this channel. Refer to Table 181 , page 280.
22	Reserved	0	Reserved.
[21:14]	WRITE_ADJ	0	This field contains a binary value, indicating the number of M3_CLK periods which the PDMA must wait after completion of a read or write access to a peripheral before evaluating the out-of-band status signals from that peripheral for another transfer. This is typically used to ensure that a posted write has fully completed to the peripheral in cases where the peripheral is running at a lower clock frequency than the PDMA. However, it may also be used to allow the PDMA to take account of internal latencies in the peripheral, where the ready status of a FIFO may not be available for a number of clock ticks after a read or write, due to internal synchronization delays, for example, within the peripheral. This applies particularly in the case of user-designed peripherals in the FPGA fabric.
[13:12]	DEST_ADDR_INC	00	This field controls the address increment at the destination end of the DMA transfer. The values have the following meanings: 00: 0 bytes 01: 1 byte 10: 2 bytes 11: 4 bytes
[11:10]	SRC_ADDR_INC	00	This field controls the address increment at the source end of the DMA transfer. The values have the following meanings: 00: 0 bytes 01: 1 byte 10: 2 bytes 11: 4 bytes
9	HI_PRIORITY	0	When asserted, this channel is treated as high priority by the arbitration state machine.
8	CLR_COMP_B	0	When asserted, clears the CH_COMP_B bit in the channel status register and the buffer status register for this buffer in this channel. This causes PDMA_INTERRUPT to negate if not being held asserted by another channel. This bit always reads back as zero.

Table 180 • CHANNEL_x_CONTROL (continued)

Bit Number	Name	Reset Value	Description
7	CLR_COMP_A	0	When asserted, clears the CH_COMP_A bit in the channel status register and the buffer status register for this buffer in this channel. This causes PDMAINTERRUPT to negate if not being held asserted by another channel. This bit always reads back as zero.
6	INTEN	0	When asserted, a DMA completion on this channel causes PDMAINTERRUPT to assert.
5	RESET	0	When asserted, resets this channel. Always reads back as zero.
4	PAUSE	0	When asserted, pauses the transfers for this channel.
[3:2]	TRANSFER_SIZE	00	This field determines the data width of each DMA transfer cycle for this DMA channel. The allowed values are: 00: Byte (8 bits) 01: Halfword (16 bits) 10: Word (32 bits) 11: Reserved
1	DIR	0	If PERIPHERAL_DMA = 1, then this bit is valid. If so, then the values of this bit have the following meanings: 0: Peripheral to memory 1: Memory to peripheral
0	PERIPHERAL_DMA	0	0: Channel is configured for memory to memory DMA. 1: Channel is configured for peripheral DMA. Based on the value of DIR, the peripheral ready signal associated with this DMA channel is interpreted as initiating transfers either from memory to the peripheral or vice-versa.

The following table gives the PERIPHERAL_SEL bits description.

Table 181 • PERIPHERAL_SEL

Bit 26	Bit 25	Bit 24	Bit 23	Function
0	0	0	0	From UART_0 receive to any MSS memory-mapped location
0	0	0	1	From any MSS memory-mapped location to UART_0 transmit
0	0	1	0	From UART_1 receive to any MSS memory-mapped location
0	0	1	1	From any MSS memory-mapped location to UART_1 transmit
0	1	0	0	From SPI_0 receive to any MSS memory-mapped location
0	1	0	1	From any MSS memory-mapped location to SPI_0 transmit
0	1	1	0	From SPI_1 receive to any MSS memory-mapped location
0	1	1	1	From any MSS memory-mapped location to SPI_1 transmit
1	0	0	0	To/from FPGA fabric peripheral on FIC_0 interface (DMAREADY_0[1])
1	0	0	1	To/from FPGA fabric peripheral on FIC_0 interface (DMAREADY_0[0])
1	0	1	0	From any MSS memory-mapped location to CAN
1	0	1	1	From CAN to any MSS memory-mapped location
1	1	0	0	To/from FPGA fabric peripheral on FIC_1 interface (DMAREADY_1[1]).
1	1	0	1	To/from FPGA fabric peripheral on FIC_1 interface (DMAREADY_1[0]).
1	1	1	0	From COMM_BLK receive to any MSS memory-mapped location
1	1	1	1	From any MSS memory-mapped location to COMM_BLK transmit

9.4.1.4 CHANNEL_x_STATUS Register Bit Definition

Table 182 • CHANNEL_x_STATUS

Bit Number	Name	Reset Value	Description
[31:3]	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
2	BUF_SEL	0	0: Buffer A is used 1: Buffer B is used
1	CH_COMP_B	0	Asserts when this channel completes its DMA. Cleared by writing to CLR_COMP_B, bit 8 in CHANNEL_x_CONTROL register for this channel. If INTEN is set for this channel, the assertion of CH_COMP_B causes PDMAINTERRUPT to assert.
0	CH_COMP_A	0	Asserts when this channel completes its DMA. Cleared by writing to CLR_COMP_A, bit 8 in CHANNEL_x_CONTROL register for this channel. If INTEN is set for this channel, the assertion of CH_COMP_A causes PDMAINTERRUPT to assert.

9.4.1.5 CHANNEL_x_BUFFER_A_SRC_ADDR Register Bit Definition

Table 183 • CHANNEL_x_BUFFER_A_SRC_ADDR

Bit Number	Name	Reset Value	Description
[31:0]	BUF_A_SRC	0	Start address from which data is to be read during the next DMA transfer cycle. If PERIPHERAL_DMA = 1 and DIR = 0, this value is not incremented from one DMA transfer cycle to the next. Otherwise, it is always incremented by an amount corresponding to the TRANSFER_SIZE for this channel.

9.4.1.6 CHANNEL_x_BUFFER_A_DST_ADDR Register Bit Definition

Table 184 • CHANNEL_x_BUFFER_A_DST_ADDR

Bit Number	Name	Reset Value	Description
[31:0]	BUF_A_DST	0	Start address from which data is to be write during the next DMA transfer cycle. If PERIPHERAL_DMA = 1 and DIR = 1, this value is not incremented from one DMA transfer cycle to the next. Otherwise, it is always incremented by an amount corresponding to the TRANSFER_SIZE for this channel.

9.4.1.7 CHANNEL_x_BUFFER_A_TRANSFER_COUNT Register Bit Definition

Table 185 • CHANNEL_x_BUFFER_A_TRANSFER_COUNT

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
[15:0]	BUF_A_COUNT	0	Number of remaining transfers to be completed between source and destination for buffer A for this channel. This field is decremented after every DMA transfer cycle. Writing a non-zero value to this register causes the DMA to start. This must be the last register written by firmware when setting up a DMA transfer.

9.4.1.8 CHANNEL_x_BUFFER_B_SRC_ADDR Register Bit Definition

Table 186 • CHANNEL_x_BUFFER_B_SRC_ADDR

Bit Number	Name	Reset Value	Description
[31:0]	BUF_B_SRC	0	Start address from which data is to be read during the next DMA transfer cycle. If PERIPHERAL_DMA = 1 and DIR = 0, this value is not incremented from one DMA transfer cycle to the next. Otherwise, it is always incremented by an amount corresponding to the TRANSFER_SIZE for this channel.

9.4.1.9 CHANNEL_x_BUFFER_B_DST_ADDR Register Bit Definition

Table 187 • CHANNEL_x_BUFFER_B_DST_ADDR

Bit Number	Name	Reset Value	Description
[31:0]	BUF_B_DST	0	Start address from which data is to be write during the next DMA transfer cycle. If PERIPHERAL_DMA = 1 and DIR = 1, this value is not incremented from one DMA transfer cycle to the next. Otherwise, it is always incremented by an amount corresponding to the TRANSFER_SIZE for this channel.

9.4.1.10 CHANNEL_x_BUFFER_B_TRANSFER_COUNT Register Bit Definition

Table 188 • CHANNEL_x_BUFFER_B_TRANSFER_COUNT

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
[15:0]	BUF_B_COUNT	0	Number of remaining transfers to be completed between source and destination for buffer B for this channel. This field is decremented after every DMA transfer cycle. Writing a non-zero value to this register causes the DMA to start. This must be the last register to be written by firmware when setting up a DMA transfer.

9.5 SYSREG Control Registers

In addition to the specific PDMA registers found in [Table 177](#), page 275, the registers found in the following table also control the behavior of the PDMA peripheral. These registers are located in the SYSREG section of the user's guide and are listed here for convenience. Refer to [System Register Block](#), page 670 for a detailed description of each register and associated bits.

Table 189 • SYSREG Control Registers

Register Name	Register Type	Flash Write Protect	Reset Source	Description
SOFT_RESET_CR	RW-P	Bit	SYSRESET_N	Soft reset control
MASTER_WEIGHT1_CR	RW-P	Register	SYSRESET_N	Configures weighted round robin master arbitration scheme for masters

10 Universal Serial Bus OTG Controller

Universal serial bus (USB) is an industry standard that defines cables, connectors, and serial communication protocol used in a bus for connection, communications, and power supply between electronic devices. SmartFusion2 SoC FPGA device contains a USB On-The-Go (OTG) controller as part of the microcontroller subsystem (MSS). SmartFusion2 USB OTG controller provides a mechanism for the USB communication between the SmartFusion2 devices and external USB host/USB device/USB OTG protocol compliant devices.

10.1 Features

- Operates as a USB host in point-to-point or multi-point communication with other USB devices
- Operates as a USB peripheral with other USB hosts
- Compliant with the USB 2.0 standard and includes OTG supplement
- Supports USB 2.0 speeds:
 - High speed (480 Mbps)
 - Full speed (12 Mbps)
- Supports session request protocol (SRP) and host negotiation protocol (HNP)
- Supports suspend and resume signaling
- Supports multipoint capabilities
- Supports four direct memory access (DMA) channels for data transfers
- Support for high bandwidth isochronous (ISO) pipe enabled endpoints
- Hardware selectable option for 8-bit/4-bit low pin interface (LPI) interface
- Supports the following two hardware interfaces to external USB physical layer (PHY):
 - UTMI+ Level 3 transceiver interface with fabric
 - ULPI link interface direct to inputs/outputs (I/Os)
- Soft connect/disconnect
- Configurable for up to 5 transmit endpoints (TX EP) and up to 5 receive endpoints (RX EP), including control endpoint (EP0)
- Offers dynamic allocation of endpoints, to maximize the number of devices supported
- Internal memory of 8 KB with support for dynamic allocation to each end point
- Performs all USB 2.0 transaction scheduling in hardware
- Supports link power management
- Single Error Correction and Dual Error Detection (SECEDED) protection on internal USB memory with the following features:
 - Generates interrupts on 1-bit or 2-bit errors; these interrupts can be masked
 - Corrects 1-bit errors
 - Counts the number of 1-bit and 2-bit errors

For more information on USB 2.0 and OTG protocol specifications, refer to the following web pages:

- www.usb.org/developers/docs/
- www.usb.org/developers/onthego/

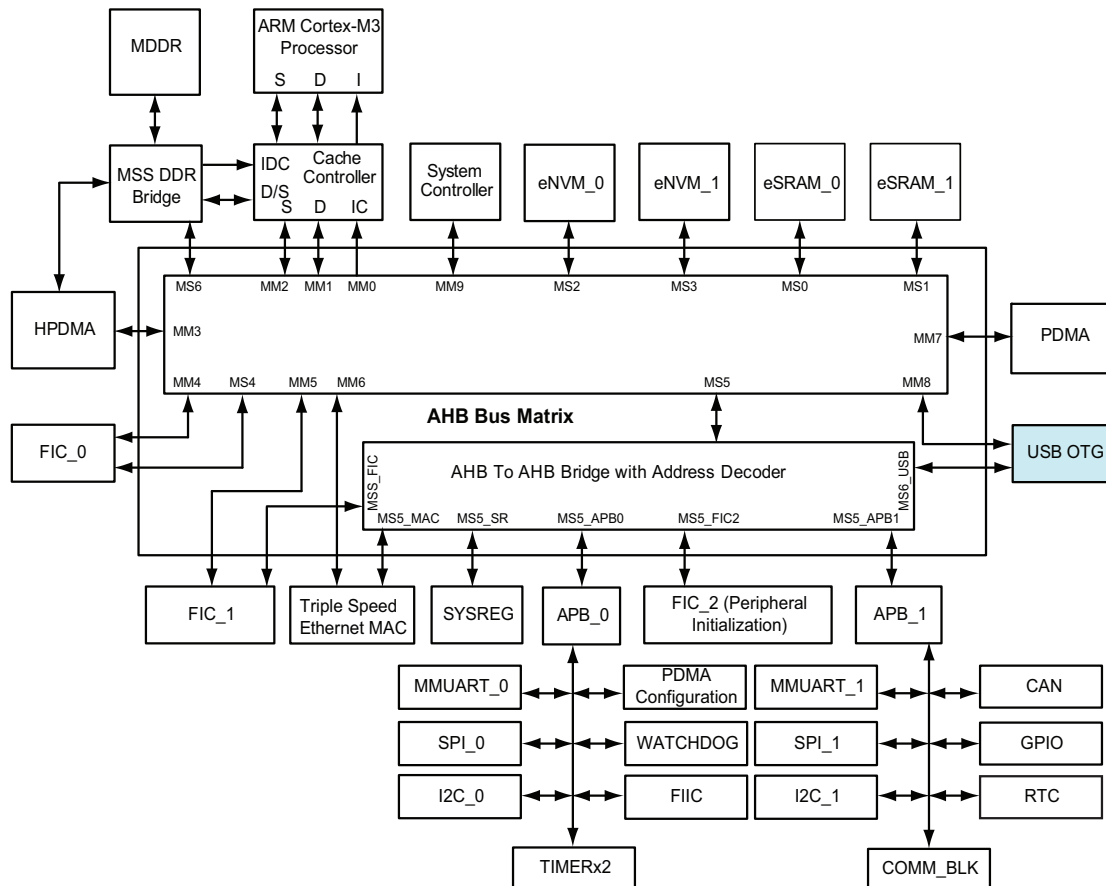
The following figure shows details of MSS. As shown in the figure, USB OTG controller can function as an AHB master for DMA data transfers and as an AHB slave for configuring the USB OTG controller from the masters ARM® Cortex® -M3 processor or from the FPGA fabric logic.

The SmartFusion2 USB OTG controller can function as:

- A high speed or a full speed peripheral USB device attached to a conventional USB host (such as a PC)
- A point-to-point or multi-point USB host
- An OTG device that can dynamically switch roles from the host and the device

In all cases (USB host, USB device, or USB OTG), SmartFusion2 USB OTG supports control, bulk, ISO, and interrupt transactions in all 3 modes.

Figure 138 • MSS Showing a USB OTG Controller



10.2 Functional Description

This section provides a detailed description of the USB OTG controller.

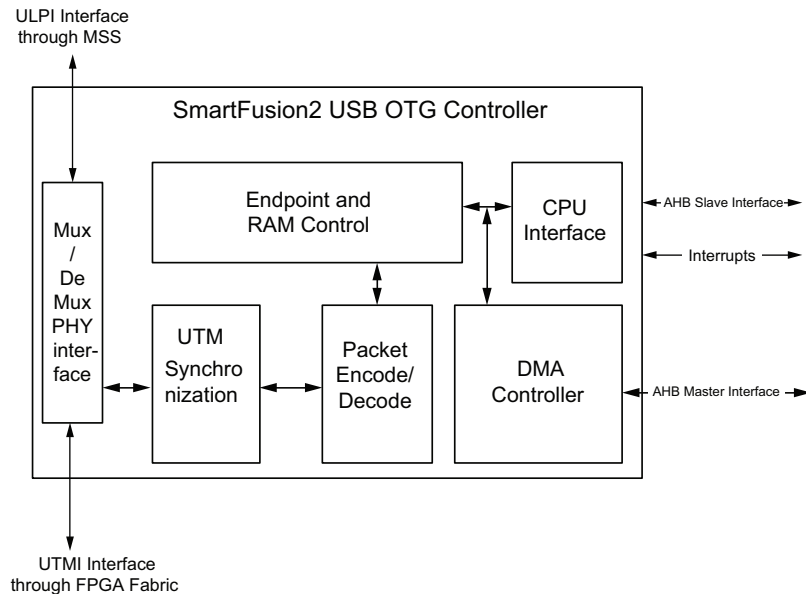
10.2.1 Architecture Overview

The following block diagram highlights the main blocks in the USB OTG controller. The USB OTG controller is interfaced through the advanced high-performance bus (AHB) matrix in the MSS. The SmartFusion2 USB OTG provides two interfaces (ULPI and UTMI) to connect to the external PHY. Following are the main component blocks in the USB OTG controller:

- AHB Master and Slave Interfaces
- CPU Interface
- Endpoints (EP) Control Logic and RAM Control Logic
- Packet Encoding, Decoding, and CRC Block

- UTM Synchronization
- PHY Interfaces

Figure 139 • USB OTG Controller in SmartFusion2



10.2.1.1 AHB Master and Slave Interfaces

The USB OTG controller functions as both AHB master and AHB slave on the AHB bus matrix. Refer to the [AHB Bus Matrix](#), page 210 for more information. The AHB master interface is used by the DMA engine, which is built into the USB controller, for data transfer between memory in the USB controller and the system memory. The AHB slave interface is used by other masters, such as the Cortex-M3 processor or Fabric masters in the FPGA fabric, to configure registers in the USB controller. The address map for the USB controller is 0x40043000-0x40043FFF.

10.2.1.2 CPU Interface

USB OTG controller sends interrupts to the Cortex-M3 processor using the CPU interface. The SmartFusion2 USB OTG controller sends interrupts for the following events:

- When packets are transmitted or received
- When the USB OTG controller enters Suspend mode
- When USB OTG resumes from Suspend mode

The CPU interface block contains the common configuration registers and the interrupt control logic for configuring the OTG controller.

10.2.1.3 Endpoints (EP) Control Logic and RAM Control Logic

These two blocks constitute buffer management for the data buffers in Host mode and in Device mode. This block manages end point buffers and their properties, called pipes, which are defined by control, bulk, interrupt and ISO data transfers. Data buffers in device mode (endpoints) and in host mode are supported by the SECDDED block, which will automatically take care of single bit error correction and dual bit error detection. This SECDDED block maintains the counters for the number of single bit corrections made and the number of detections of dual bit errors. The SECDDED block is provided with interrupt generation logic. If enabled, this block will generate the corresponding interrupts to the Cortex-M3 processor in SmartFusion2.

10.2.1.4 Packet Encoding, Decoding, and CRC Block

This block generates the CRC for packets to be transmitted and checks the CRC on received packets. This block generates the headers for the packets to be transmitted and decodes the headers on received packets. There is a CRC 16-bit for the data packets and a 5-bit CRC for control and status packets.

10.2.1.5 UTM Synchronization

The role of the UTM synchronize block is to synchronize between the transceiver macrocell 60 MHz clock domain and the SmartFusion2 USB OTG controller's system clock. This allows the rest of the USB OTG controller to run at the desired system clock. This block also performs the high speed detection handshaking and handles the host negotiation protocol (HNP) and the session request protocol of the OTG specification in point-to-point communications with another USB OTG device.

10.2.1.6 PHY Interfaces

The USB controller supports both universal transceiver macrocell interface plus (UTMI+) and universal low pin count interface (ULPI) at the link side. For ULPI interface I/Os are routed through the MSS onto multi standard I/Os (MSIOs) and for UTMI, I/Os are routed through the FPGA fabric. The built-in multiplexer (MUX) and de-multiplexer (DeMUX) logic allows the selection of UTMI or ULPI through the Libero SoC software.

Note: The SmartFusion2 devices support OTG PHY but do not support the TI NON-OTG PHY.

10.2.2 USB OTG Controller Interface Signals

This section describes the ULPI and UTMI+ fabric interfaces.

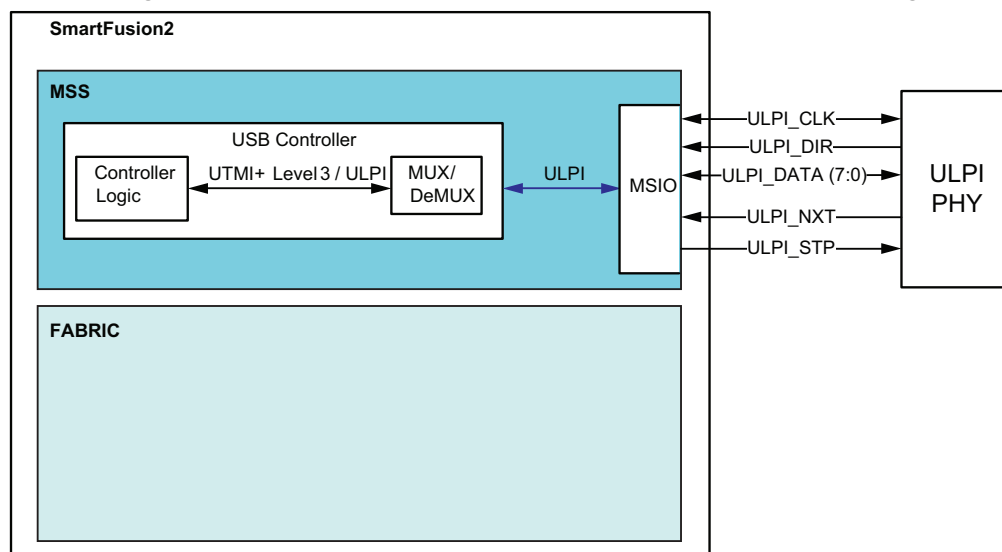
10.2.2.1 ULPI (UTMI+ Low Pin Interface) I/O Interface

The SmartFusion2 USB OTG controller communicates with the external ULPI PHY device using this interface. As shown in the following figure, the ULPI interface is routed through the MSIO ports. These I/Os are dedicated to the USB ULPI interface only. When the USB OTG controller is selected during configuration, these I/Os are not multiplexed with other peripherals.

For interfacing with ULPI PHY, the USB MSIO signals are connected to four separate mutually exclusive I/O groups: USBA, USBB, USBC, and USBD I/O groups. In M2S050 devices, only the USBD I/O group is available; whereas in the M2S025 and M2S010, only the USBA, USBB, and USBC I/O groups are available.

Table 190 • ULPI Interface Signals at SmartFusion2 External I/Os

Signal Name	Direction	Description
ULPI_DATA[7:0]	In/Out	ULPI input data bus to ULPI link wrapper
ULPI_DIR	In	Controls the direction of the data bus. The PHY must drive this signal high when it has the data to be transferred. Otherwise, the PHY should drive this signal low.
ULPI_STP	Out	Data end control, driven high for one XCLK cycle to indicate the end of a transmit operation. It can also be used to stop the current receive operation. Asynchronous path from DIR.
ULPI_NXT	In	Data control, driven high by the PHY to throttle all data types except the interrupt data and the results of register reads.
ULPI_XCLK	In	Transceiver macrocell clock; 60 MHz

Figure 140 • Block Diagram for Connections between USB Controller and ULPI PHY through MSS

The ULPI interface is connected to the IOs of the device up to four separate sets of IOs. The function of each IO in these sets is shared with another MSS function. These IO groups allow flexibility to the user in deciding which MSS peripherals to connect to the IOs. The four USB IO groups are referred to as USB_A, USB_B, USB_C and USB_D. User can try each USB IO group and see which pins cause conflict and iterate through different assignments with the conflicting peripherals and GPIO. The availability of these USB IO groups are device-dependent, as specified in the following table.

Table 191 • USB IO Group Availability

Device	USB_A	USB_B	USB_C	USB_D
M2S005	No	Yes	No	No
M2S010	Yes	Yes	Yes	No
M2S025	Yes	Yes	Yes	No
M2S050 / 060 / 090 / 150	Yes	Yes	Yes	Yes

For more information on USB IO group, refer PPAT documents available at <https://www.microsemi.com/product-directory/socfpgas/1692-smartfusion2#documentation> (PPAT documents are grouped under the section Pinout/Packaging in the above mentioned site).

10.2.2.2 UTMI+ (USB 2.0 Transceiver Macrocell Interface+) Interface

This is the external interface connecting the SmartFusion2 USB OTG controller to an off-chip UTMI PHY device. For UTMI interface, all the interface signals are routed through the FPGA fabric on to the MSIOs.

Table 192 • UTMI+Interface Signals at Fabric Interface in SmartFusion2 Device

Signal Name	Direction	Description
UTMI_SUSPENDM	Out	Indicates asynchronous Suspend mode (derived from signals from both CLK and XCLK flip-flops). When enabled through bit 0 of the Power register, goes low when the device is in Suspend mode. Otherwise high (intended to drive a UTMI PHY).

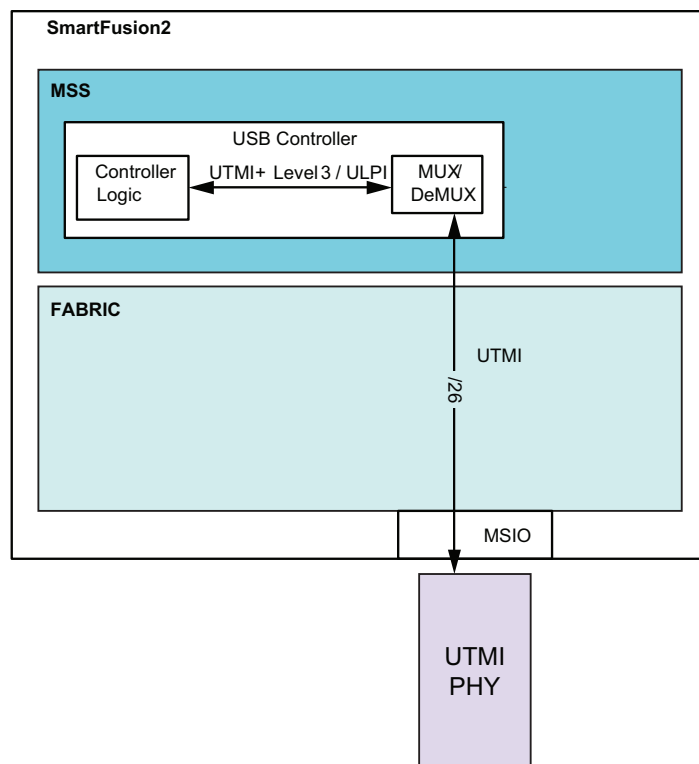
Table 192 • UTMI+Interface Signals at Fabric Interface in SmartFusion2 Device (continued)

Signal Name	Direction	Description
UTMI_LINESTATE[1:0]	In	Shows the current state of single-ended receivers. LINESTATE[0] reflects the state of D+; LINESTATE[1] reflects state of D-. 00: SE0 01: J State 10: K State 11: SE1
UTMI_OPMODE[1:0]	Out	Selects Operating mode 00: Normal operation 01: Non-driving 10: Bit stuffing and NRZI encoding disabled 11: Reserved
UTMI_XDATAIN[7:0]	In	Received data
UTMI_XDATAOUT[7:0]	Out	Data to be transmitted
UTMI_TXVALID	Out	Transmit data valid. Indicates that valid data has been transmitted.
UTMI_TXREADY	In	Transmit data ready. Indicates that the transmitter requires data.
UTMI_RXVALID	In	Receive data valid. Indicates that valid data has been received.
UTMI_RXACTIVE	In	Indicates that a valid packet is being received.
UTMI_RXERROR	In	Indicates that the received packet is about to be aborted due to an error.
UTMI_XCVRSEL[1:0]	Out	Transceiver select 00: HS transceiver 01: FS transceiver 10: LS transceiver 11: FS transceiver, LS packet
UTMI_TERMSEL	Out	Termination select. When 0, high speed termination is enabled; when 1, full speed termination is enabled. May be used to switch the pull-up resistor on D+
UTMI_VBUSVALID	In	Compares VBus to selected VBus valid threshold (required to be between 4.4 V and 4.75 V) 1: Above the VBus valid threshold 0: Below the VBus valid threshold
UTMI_AVALID	In	Compares VBus to session valid threshold for a B device (required to be between 0.8 V and 2 V) 1: Above the session valid threshold 0: Below the session valid threshold
UTMI_SESEND	In	Compares VBus to session end threshold (required to be between 0.2 V and 0.8 V) 0: Above the session end threshold 1: Below the session end threshold
UTMI_SESEND	In	Compares VBus to session end threshold (required to be between 0.2 V and 0.8 V) 0: Above the session end threshold 1: Below the session end threshold
UTMI_DRVVBUS	Out	Enables VBus power (used when the USB controller operates as an A device)
UTMI_CHRGVBUS	Out	Charges VBus (used during session request when the USB controller operates as a B device)

Table 192 • UTMI+Interface Signals at Fabric Interface in SmartFusion2 Device (continued)

Signal Name	Direction	Description
UTMI_DISCHRGVBUS	Out	Discharges VBus (used by B devices to ensure that the VBus is low enough before starting a session request protocol (SRP))
UTMI_HOSTDISCON	In	Host mode only; must be asserted when a high speed disconnect occurs (in accordance with the UTMI+ specification). Full/low speed connections are monitored through the LINESTATE signal.
UTMI_DPPULLDOWN	Out	Enables a pull-down resistor within the transceiver on the D+ line – Low when the USB controller is operating as a peripheral – High when USB controller is operating as a host
UTMI_DMPULLDOWN	Out	Enables a pull-down resistor within the transceiver on the D– line. Needs to be high, when the USB controller is used for point-to-point communications.
UTMI_IDDIG	In	Indicates USB controller connector type. High = B-type, Low = A-type.
UTMI_IDPULLUP	Out	Enables for IDDIG signal generation
UTMI_VSTATUS[7:0]	In	PHY status data; 8-bit wide as per UTMI+ specifications
UTMI_VCONTROL[3:0]	Out	PHY control data; 8-bit wide as per UTMI+ specifications
UTMI_VCONTROLLOADM	Out	Active low signal; asserted when new control information is required to be read – if implemented
UTMI_RXVALIDH	In	Tied permanently low at the fabric interface

UTMI+Level3 signals are routed through the FPGA fabric onto MSIOs.

Figure 141 • Block Diagram for Connections Between USB Controller and UTMI PHY through FPGA Fabric

Note: The FPGA fabric design should implement a 'AND' logic between UTMI_TXVALID and UTMI_TXREADY for proper data transmission.

For more information on USB 2.0 PHY interfaces refer to the following links:

- www.ulpi.org
- www.ulpi.org/ulpi_whitepaper_v2.pdf
- www.ulpi.org/documents.html
- www.ulpi.org/utmiplus_whitepaper.pdf
- www.usb.org/developers

10.2.3 USB OTG Controller Operations

This section describes the following:

- Modes of USB OTG Controller Operation
- USB OTG Controller: Reset Operations
- USB OTG Controller: Suspend/Resume Operations
- USB OTG Controller: Connect/Disconnect Operations

10.2.3.1 Modes of USB OTG Controller Operation

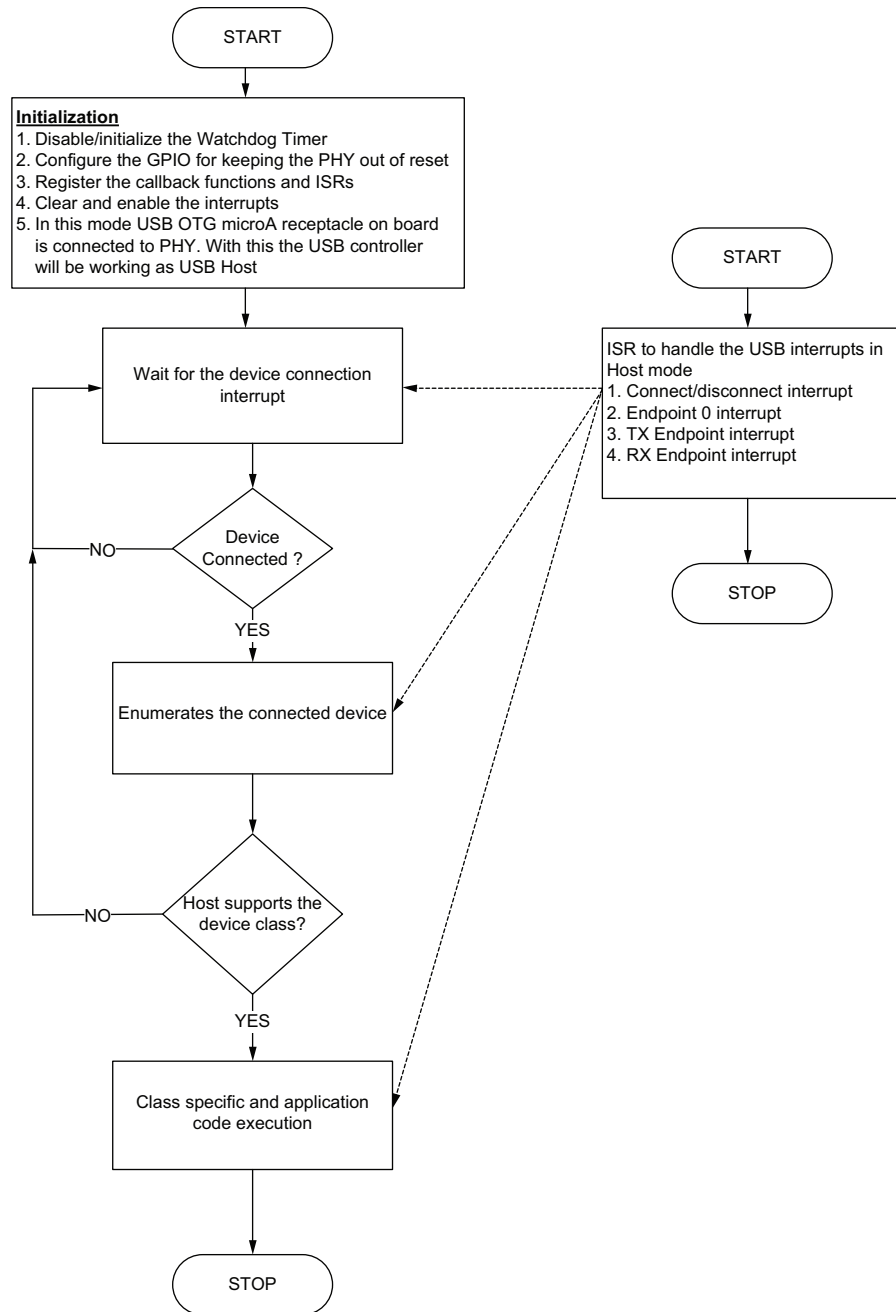
The USB OTG Controller in SmartFusion2 device can be used in the following three modes:

10.2.3.1.1 USB Host Mode

In this mode, the USB OTG controller acts in a USB host function. As the USB protocol is host driven, the USB host is completely responsible for all the transactions in the bus. In this mode, the USB OTG controller enumerates the external device that is connected. Based on the USB firmware, class drivers, and application code implemented in the SmartFusion2 device; if the connected device USB class is supported, the SmartFusion2 USB OTG controller exchanges the data with the connected device, as per the application requirement of the USB function.

The USB multi-point capability of USB controller is associated with a range of registers (Table 297, page 358). These registers are needed to record the allocation of device functions to individual controller endpoints, and device function characteristics such as; endpoint number, operating speed, and transaction type on an endpoint-by-endpoint basis. Although principally associated with the use of the core as the host to a number of devices, these registers are also required to be set when the core is used as the host for a single device.

The following figure shows the flow chart of the overall operation in the USB Host mode.

Figure 142 • Basic USB Flow Diagram when USB Controller is in Host Mode

10.2.3.1.2 USB Device/Peripheral Mode

In this mode, the USB OTG controller acts as a USB device, and device functionality are defined by the standard- class- specific or vendor- specific- class firmware and application implementations. In this mode, the USB controller; along with associated firmware, class drivers, and application code, responds to the USB commands sent by the USB host that is connected.

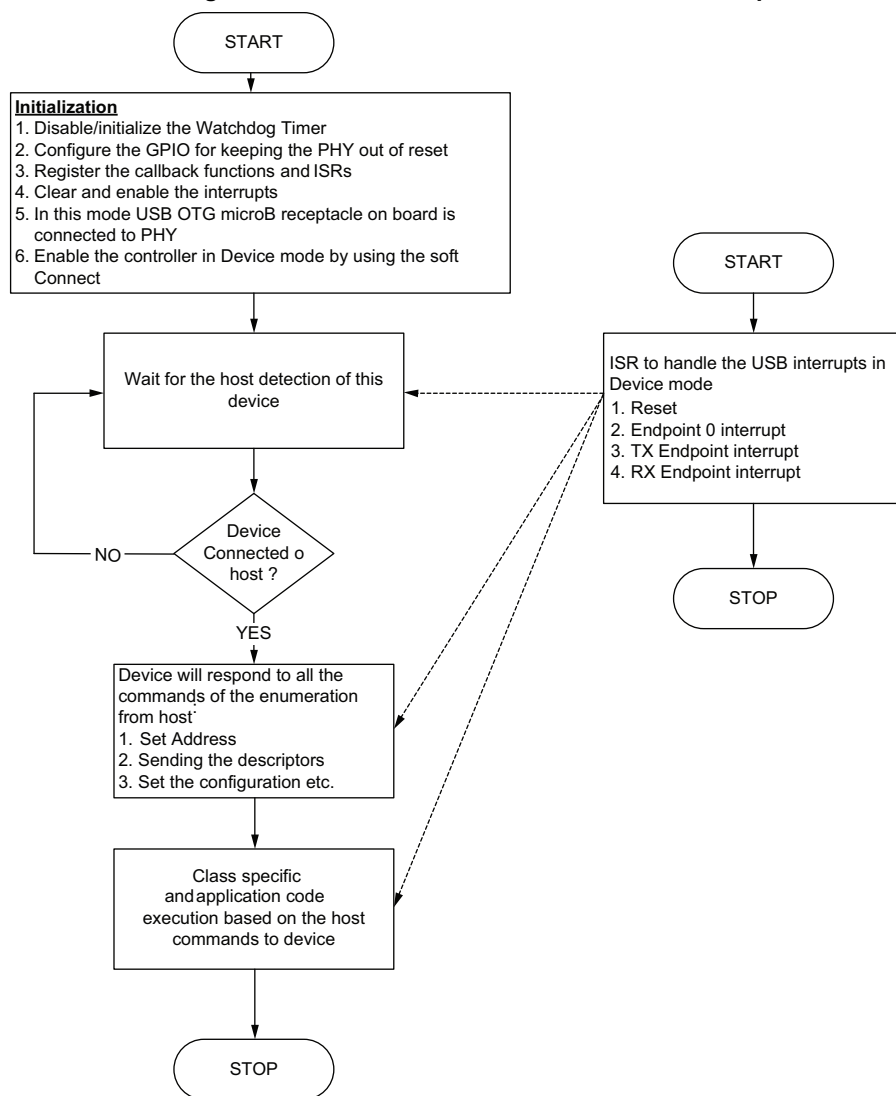
The USB controller allows its connection to the USB bus to be controlled by the software. When the USB controller operates in Peripheral mode, the UTMI+-compliant PHY that is used alongside the USB controller, can be switched between normal mode and non-driving mode by setting/clearing the Soft Conn bit in [POWER_REG \(0x40043001\)](#).

When the Soft Conn bit is set to 1, the PHY is placed in its normal mode and the D+/D- lines of the USB bus are enabled. When this feature is enabled and the Soft Conn bit is zero, the PHY is put into non-driving mode (OPMODE[1:0] set to 01) and D+ and D- are tristated. The USB controller then appears to other devices on the USB bus to have been disconnected.

After hardware reset, Soft Conn is cleared to 0. The USB controller therefore appears disconnected until the software has set Soft Conn to 1. The application software can then choose when to set the PHY into its normal mode. Systems with a lengthy initialization procedure may use this to ensure that initialization is completed and the system is ready to perform enumeration before connecting to the USB.

The following figure shows the flow chart of the overall operation in the USB Device mode.

Figure 143 • Basic USB Flow Diagram when USB Controller is in USB Device/Peripheral Mode



10.2.3.1.3 USB OTG Mode (Dual Role)

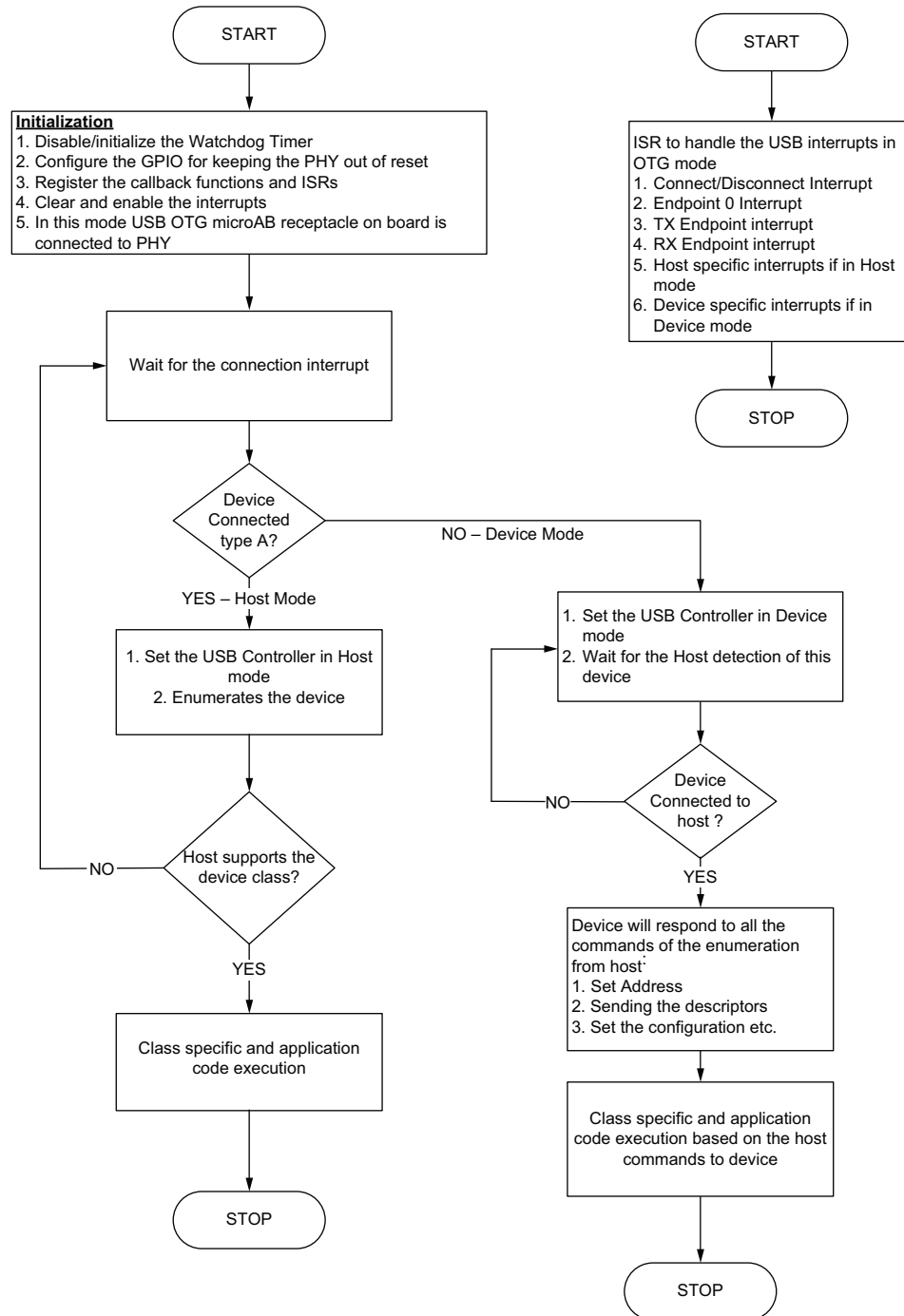
In this mode, the SmartFusion2 USB OTG controller must be configured in OTG mode by using the USB controller register. Based on the type of the plug connected to the PHY through the USB OTG receptacle, the controller plays a role as either the USB device or the USB host. If the plug/device with type microA is connected, the corresponding interrupt generated to the USB controller and the firmware configures the USB OTG controller into the USB Host mode. If the plug/device with microB is connected, the corresponding USB interrupt will be generated to the Cortex-M3 processor and the firmware ISR will configure the USB OTG controller into the USB Device mode.

In OTG mode, whether the USB controller expects to behave as a host or as a peripheral, depends on the way the devices are cabled together. Following list explains the USB cable, how the USB OTG controller decides the role (device/host):

- **USB Cable end types:** Each USB cable has an A / microA / miniA end and a B / microB / miniB end.
- **Entering into host mode:** If the micro A end of the cable is plugged into the SmartFusion2 device through the external PHY, the USB controller will take the role of the host and go into Host mode.
- **Entering into device mode:** If the micro B end of the cable is plugged in, the USB controller will go instead into Peripheral mode and Host mode bit will be set to 0.
- **Changing the role without swapping the cable ends:** Where the USB controller is connected to a single device that contains a dual-role controller, signaling can be used to switch the roles of the two devices—with no need to switch the cable between the devices. The conditions under which the USB controller may switch between a peripheral role and a host role are explained in the OTG specification.

The following figure shows the flow chart of the overall operation in the USB OTG (dual role) mode.

Figure 144 • Basic USB Flow Diagram when USB Controller is in OTG Mode



10.2.3.2 USB OTG Controller: Reset Operations

10.2.3.2.1 Peripheral Mode

When the USB controller is acting as a peripheral and a reset condition is detected on the USB, the device performs the following actions:

1. Sets Func Addr in **FADDR_REG(0x40043000)** to 0
2. Sets Selected Endpoint in **INDEX_REG (0x4004300E)** to 0

3. Flushes all endpoint FIFOs
4. Clears all control/status registers
5. Enables all endpoint interrupts
6. Generates a reset interrupt

If the HS Enab bit in **POWER_REG (0x40043001)** is set, the USB controller tries to negotiate for high speed operation.

Whether or not the high speed operation is selected, is indicated by the HS mode bit in **POWER_REG**.

When the application software driving the USB controller receives a reset interrupt, it should close any open pipes and wait for the bus enumeration to begin.

10.2.3.2.2 Host Mode

If the reset bit in **POWER_REG** is set while the USB controller is in Host mode, the USB controller generates reset signaling on the bus. If the HS Enab bit in **POWER_REG** is set, it tries to negotiate for a high speed operation.

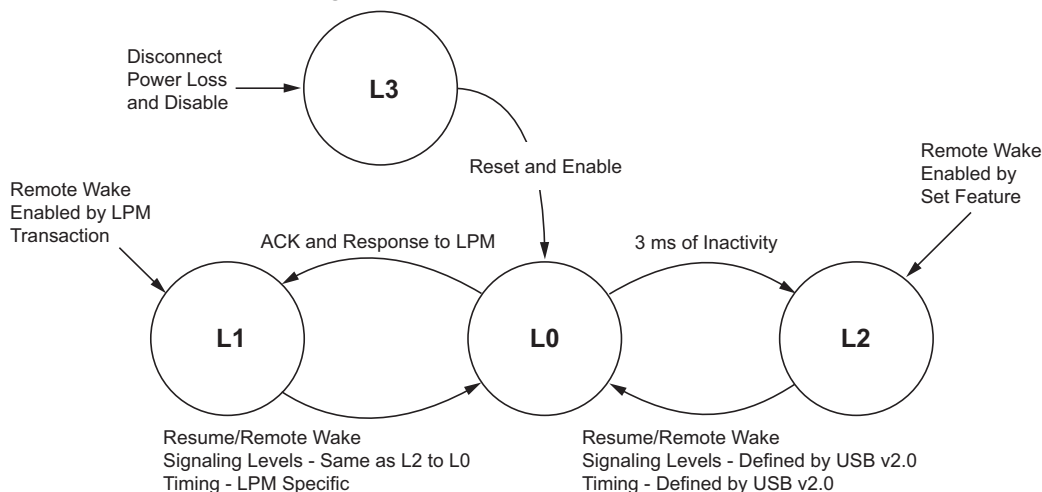
The Cortex-M3 processor or fabric master should keep the reset bit set for at least 20 ms to ensure the correct resetting of the target device.

After the Cortex-M3 processor or fabric master clears the bit, the USB controller starts its frame counter and transaction scheduler. Whether a high speed operation is selected, is indicated by the HS mode bit of **POWER_REG (0x40043001)**.

10.2.3.3 USB OTG Controller: Suspend/Resume Operations

With the introduction of link power management (LPM), there are two basic methods for the USB controller to be suspended and resumed. These two methods are demonstrated in the basic LPM transaction diagram as shown in the following figure.

Figure 145 • LPM State Transition Diagram



The procedure in which the controller is suspended and resumed depends on whether the controller is operating as a device (peripheral) or as a host, and the method of suspend desired.

10.2.3.3.1 Suspend/Resume by Inactivity on the USB Bus (L0 to L2 State Transition)

Suspend/Resume when Operating as a Peripheral

- **Entry into Suspend Mode:** When operating as a peripheral, the USB controller monitors activity on the USB and when no activity has occurred for 3 ms, it goes into Suspend mode. If the Suspend interrupt **USB_IRQ_EN_REG (0x4004300B)** has been enabled, an interrupt is generated at this time. The **SUSPENDM** output also goes low (if enabled). At this point, the **POWERDWN** signal is also asserted to indicate that the application may save power by stopping **CLK**.

POWERDWN then remains asserted until power is removed from the bus (indicating that the device has been disconnected); or resume signaling; or reset signaling is detected on the bus.

- **Sending Resume Signaling:** When resume signaling occurs on the bus, first CLK must be restarted, if necessary. The USB controller then automatically exits in Suspend mode. If the resume interrupt is enabled, an interrupt is generated.
- **Initiating a Remote Wake-Up:** If a remote wake-up is initiated by the software while the USB controller is in Suspend mode, it should write to **POWER_REG (0x40043001)** to set the resume bit to '1'.

Note: If CLK has been stopped, it must be restarted before this write can occur. The software should leave this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0. By this time the hub should have taken over driving and resume signaling on the USB. Resume interrupt is not generated when the software initiates a remote wake-up.

Suspend/Resume When Operating as a Host

- **Entry into Suspend mode:** When operating as a host, the USB controller can be prompted to go into Suspend mode by setting the Suspend mode bit in **POWER_REG (0x40043001)**. When this bit is set, the USB controller completes the current transaction, then stops the transaction scheduler and frame counter. No further transactions are started and no SOF packet is generated. If the Enable SuspendM bit in **POWER_REG** is set, the UTMI+ PHY goes into Low power mode; when the USB controller goes into Suspend mode and stops XCLK.
- **Sending Resume Signaling:** When the application requires the USB controller to leave Suspend mode, it must clear the Suspend mode bit and set the resume bit in the **POWER_REG**, and leave it set for 20 ms. while the resume bit is high, the USB controller generates resume signaling on the bus. After 20 ms, the Cortex-M3 processor or fabric master should clear the resume bit, at which point the frame counter and transaction scheduler are started.
- **Responding to Remote Wake-Up:** If resume signaling is detected from the target while the USB controller is in Suspend mode, the UTMI+ PHY is brought out of Low power mode and restarts XCLK. The USB controller then exits Suspend mode and automatically sets the Resume bit in **POWER_REG** to '1' to take over generating the resume signaling from the target. If the resume interrupt is enabled, an interrupt is generated.

10.2.3.3.2 Suspend/Resume by an LPM Transaction (L0 to L1 State Transition)

Suspend/Resume When Operating as a Peripheral

- **Entry into Suspend mode:** When operating as a peripheral, the USB controller never initiates an LPM suspend (transition from the L0 state to the L1 state). Rather, the USB controller only suspends at the request of the host. However, for this to occur, the LPM feature must be enabled by setting up **LPM_CTRL_REG (0x40043362)(Peripheral)** appropriately. The LPMEN field in **LPM_CTRL_REG** is used to enable the extended and LPM transactions. The LPMXMT field in **LPM_CTRL_REG** is used to instruct the hardware that it is ready to suspend and to respond to the next LPM transaction with an ACK. In this case, the USB controller responds to the next LPM transaction with an ACK, if all other conditions are met. The response to an LPM transaction by the USB controller is summarized in the following table.

Table 193 • Response to LPM Transaction as Peripheral

LPMXMT	LPMEN	Data Pending (Data Resides in Transmit FIFOs)	Response to the Next LPM transaction
1'b0	00	Don't Care	Timeout
1'b0	10		
1'b1	00		
1'b1	10		
1'b0	01	Don't Care	STALL
1'b1			

Table 193 • Response to LPM Transaction as Peripheral (continued)

LPMXMT	LPMEN	Data Pending (Data Resides in Transmit FIFOs)	Response to the Next LPM transaction
1'b0	11	Don't Care	NYET
1'b1		Yes	NYET
1'b1		No	ACK

For all cases shown in the preceding table in which the USB controller responds (no timeout occurs), an LPM interrupt is generated in LPM_INTR_REG (Table 308, page 368).

Note: The USB controller responds with an ACK only if there is no data pending in any of the transmit endpoint FIFOs. If there is data pending, the USB controller responds with a NYET.

Once an LPM transaction is successfully received, three events occur:

1. **LPM_ATTR_REG (0x40043360)** is updated with values received in the LPM transaction. The fields in this register are as follows:
 - **LinkState:** This field tells the USB controller what state to transition to. The only valid value for this field is 4'h1, indicating that the core should suspend. For any other value, the USB controller responds with a STALL, and the appropriate interrupt will be generated. However, in this case the LPM_ATTR_REG is updated so that software can observe the non-compliant LPM packet payload. In addition, the LPMERR interrupt set in **LPM_INTR_REG (0x40043364)(Peripheral Mode)** is also generated, informing software of the non-compliant LPM transaction.
 - **HIRD:** This field let the USB controller know the minimum duration that the host drives resume signaling on the bus. This field represents a resume duration range from 50 μ s to 1200 μ s. This value may be different for subsequent LPM transactions.
 - **RmtWak:** This is a 1-bit field indicating whether the remote wake-up by the USB controller is allowed. This bit applies to the current suspend/resume cycle only. This bit may be different for subsequent LPM transactions. This bit should not supersede the wake-up capability that was previously negotiated on enumeration of the USB controller.
2. The USB controller suspends 9 μ s after transmitting the ACK. Resume signaling can be driven by the host or the USB controller 50 μ s after this event. During this 9 μ s interval, the host may continue to transmit the LPM transaction. The USB controller responds with an ACK in this case, regardless of the LPMXMT value in **LPM_CTRL_REG (0x40043362)(Peripheral)**.
3. An interrupt is generated, informing software of the response (an ACK in this case). An ACK response is the indication to the software that the USB controller has suspended.

Since the primary purpose of LPM is to save power, the software will read the **LPM_ATTR_REG (0x40043360)** to determine the attributes of the suspend mode. The software must make a determination based on these attributes whether additional power savings in the system can be exploited.

Note: If the host initiates resume signaling, the USB controller is required to respond to the packet transmissions within the time specified by HIRD + 10 μ s.

- **Resume Signaling Occurs on the Bus:** When the host resumes the bus, it drives resume signaling for a minimum time specified by the HIRD field in **LPM_ATTR_REG (0x40043360)**. The USB controller must be able to respond to traffic within the time HIRD + 10 μ s. The USB controller transitions to a normal operating state automatically and a resume interrupt is generated in **LPM_INTR_REG (0x40043364)(Peripheral Mode)**. However, for this to occur, the inputs CLK and XCLK must be available. To facilitate the resume timing requirement, an additional feature, LPMNAK, is provided in **LPM_CTRL_REG (0x40043362)(Peripheral)**. If LPMNAK is set to 1, all endpoints respond to any transaction (other than an LPM) with a NAK. This bit only takes effect after the USB controller has LPM suspended. Typically, this bit would be asserted when the LPMXMT field is also asserted. Using this feature may simplify the resume timing requirement because only XCLK is needed for the USB controller to respond (with a NAK) to traffic. The software can continue to restore the system to normal operation while the USB controller responds to all transactions with a NAK. After the system has been completely restored, the software can then clear the LPMNAK field in LPM_CTRL_REG.

- **Initiating Remote Wake-Up:** If the software wants to initiate a remote wake-up while the USB controller is in Suspend mode, it should write a 1 to the LPMRES bit in [LPM_CTRL_REG \(0x40043362\)\(Peripheral\)](#). This bit is self-clearing. Writing a 1 causes resume signaling to be driven on the bus for 50 μ s. The host responds by driving resume signaling for 60 μ s to 990 μ s. 10 μ s after the host stops driving resume, the USB controller transits to its normal operational state and is ready for packet transmission. A resume interrupt is generated in [LPM_INTR_REG \(0x40043364\)\(Peripheral Mode\)](#).

Suspend/Resume When Operating as a Host

- **Entry into Suspend mode:** When operating as a host, the controller initiates an LPM suspend (transition from the L0 state to the L1 state) by initiating the following LPM transaction:
 1. Software sets-up the desired attributes of the Suspend mode in [LPM_ATTR_REG \(0x40043360\)](#). Enables remote wake-up and a large HIRD gives the peripheral more opportunity to conserve power.
 2. All LPM interrupts must be enabled in [LPM_INTR_EN_REG \(0x40043363\)](#).
 3. Software should initiate the transaction by writing a 0x01 to [LPM_CTRL_REG \(0x40043362\) \(Host\)](#).
 4. An interrupt is generated to inform software of the response to the LPM transaction. If an ACK is received, the controller will be suspended automatically within 8 μ s. This is the indication that the controller is suspended.
 5. If the response from the device has a bit stuff error or a PID error, an LPMERR interrupt ([LPM_INTR_REG \(0x40043364\) \(Host Mode\)](#)) is generated. The hardware immediately attempts the LPM transaction for two more times. The device is not suspended for 8 μ s after the initial LPM, so it will be able to respond to either of these subsequent LPM transactions. If an LPM timeout occurs three times, the LPMNC and the LPMERR interrupts are set ([LPM_INTR_REG](#)). At this time, software is unaware of the device state and must deduce it by other means.
- **Sending Resume Signaling:** Resume signaling should be generated by the software as follows:
 1. All LPM interrupts should be enabled in [LPM_INTR_EN_REG \(0x40043363\)](#).
 2. Software should write the LPMRES bit in [LPM_CTRL_REG \(0x40043362\) \(Host\)](#). This bit is self clearing. This causes resume signaling to be generated on the bus for the time that is currently specified in the HIRD field in [LPM_ATTR_REG \(0x40043360\)](#). Hardware assumes that this value was used in the last LPM transaction that caused the suspend mode.
 3. After HIRD + 10 μ s, the controller transitions to its normal operational state and is ready for packet transmission. A resume interrupt is generated in [LPM_INTR_REG](#).

Note: Prior to resuming, the software must ensure that the system is completely restored from a low- power state and that the inputs CLK and XCLK are available.

- **Responding to Remote Wake-Up:** If the remote wake-up feature is enabled in the LPM transaction that caused the Suspend mode, the device may drive resume signaling on the bus. When this occurs, the device drives the resume signaling bus for 50 μ s. The controller immediately begins driving resume signaling on the bus and will do so for 60 μ s. 10 μ s after completion of the resume signaling, the controller transitions to its normal operating state and is ready for packet transmission. At this time, the resume interrupt is generated in [LPM_INTR_REG](#).

10.2.3.4 USB OTG Controller: Connect/Disconnect Operations

The particular behavior related to connecting and disconnecting the USB controller concerns its use in Host mode or Peripheral mode in peer-to-peer communications.

10.2.3.4.1 Host Mode

Where the USB controller is operating in Host mode, the Cortex-M3 processor or fabric master starts the session by setting the session bit of [DEV_CTRL_REG \(0x40043060\)](#). Power is then applied to VBus and the core waits for a device to be connected.

When a device is detected, a connect interrupt is generated (the Conn bit in [USB_IRQ_REG \(0x4004300A\)](#), goes high). The speed of the device that has been connected can be determined by reading [DEV_CTRL_REG \(0x40043060\)](#), where the FSDev bit will be high for a high speed/full speed device and the LSDev bit will be high for a low speed device. The Cortex-M3 processor or fabric master should then reset the device. If both FSDev and HS Enab ([POWER_REG \(0x40043001\)](#)) are set, the USB controller will try to negotiate for high speed operation. Whether this is successful, is indicated by the HS mode bit ([POWER_REG \(0x40043001\)](#)).

The Cortex-M3 processor or fabric master should keep the Reset bit set for 20 ms to ensure that the

target is reset. It can then begin device enumeration.

If the device is disconnected while a session is in progress, a disconnect interrupt is generated (the DisCon bit in `USB_IRQ_REG (0x4004300A)`, goes High).

10.2.3.4.2 Peripheral Mode

Where the USB controller is operating in Peripheral mode, no interrupt is generated when the device is connected to the host. However, a disconnect interrupt (`USB_IRQ_REG (0x4004300A)`) is generated when the host terminates a session.

10.3 How to Use USB OTG Controller

Microsemi recommends the following flow for configuring the USB OTG controller.

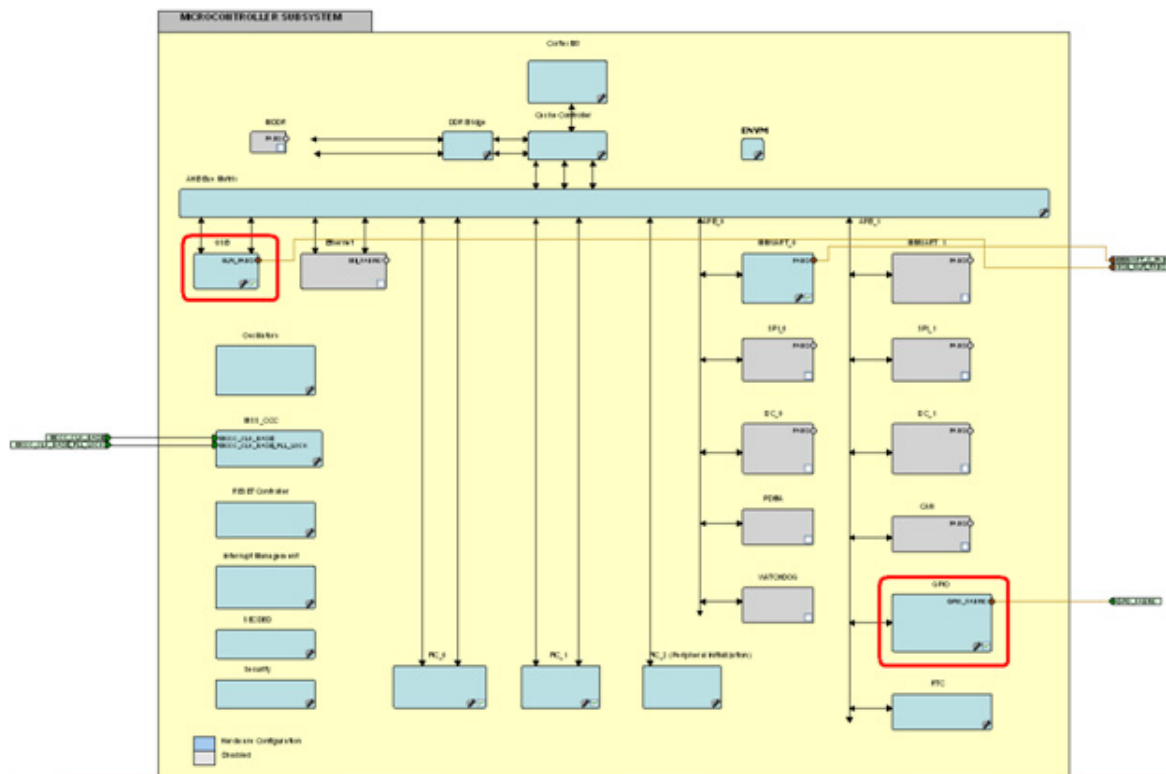
- Instantiate the SmartFusion2 MSS component into the Libero SoC project
- Configure (enable/disable) the SmartFusion2 MSS components as per the application needs using the MSS configurator
- Configure the USB OTG controller as explained in the following sections

Note: The MSS USB does not support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for more information.

10.3.1 Libero Settings for USB OTG Configuration

USB OTG controller is configured using the USB macro available in the Libero SoC design software. Using the USB macro settings options, the USB controller interface can be configured to either ULPI or UTMI interface. For external USB PHY (ULPI or UTMI) reset, Microsemi recommends to configure GPIO using the GPIO macro in the Libero SoC design software. The following figure highlights these two mandatory blocks for the USB controller configuration in the applications.

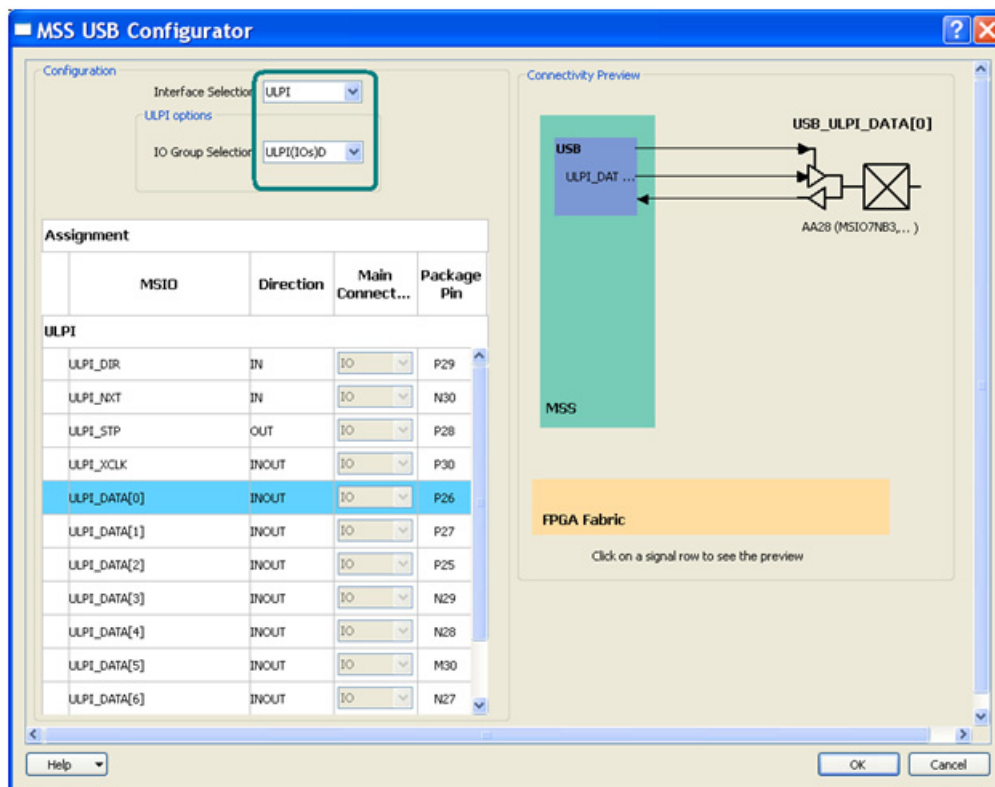
Figure 146 • MSS Configurator with USB and GPIO Macros Enabled



10.3.1.1 ULPI Interface Settings

For configuring the USB OTG controller to use with the ULPI PHY, the interface option can be selected as shown in the following figure. For interfacing the USB OTG controller with ULPI PHY, the USB MSIO signals are connected to four separate mutually exclusive I/O groups: USBA, USBB, USBC, and USBD I/O. In the M2S050 devices, only the USBD I/O group is available. Where in the M2S025 and the M2S010, only the USBA, USBB, and USBC I/O groups are available. Based on the SmartFusion2 device selected, the I/O groups can be selected from the USB configurator.

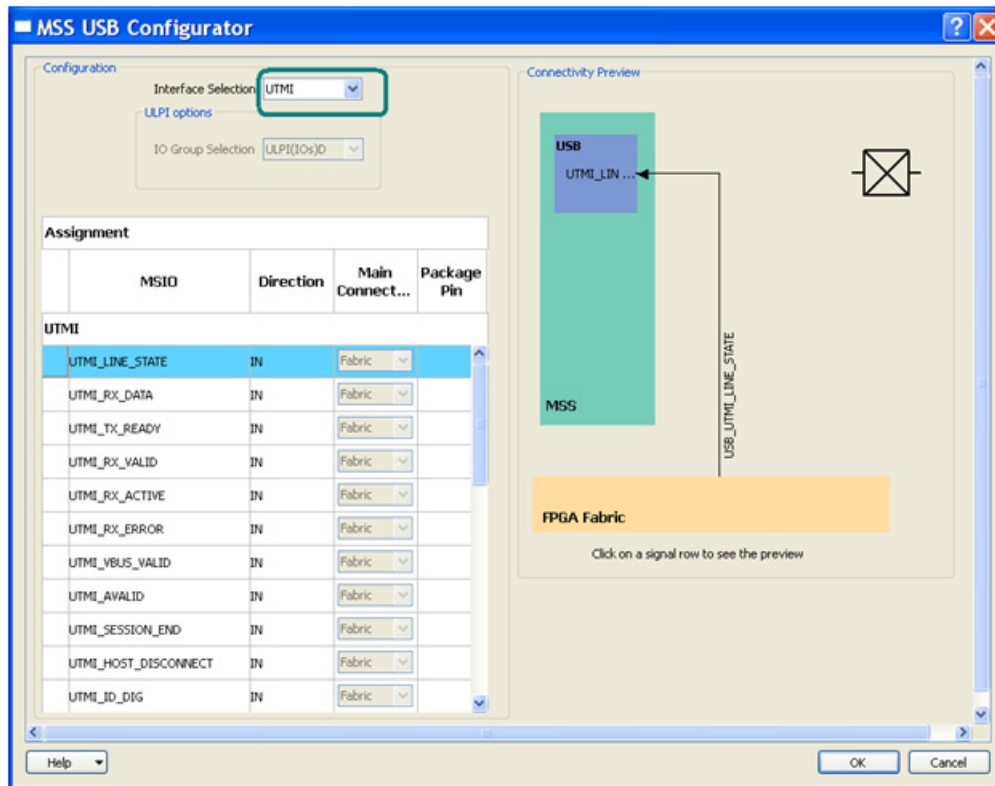
Figure 147 • MSS USB Configurator with ULPI Interface Settings



10.3.1.2 UTMI Interface Settings

If UTMI PHY is in use, then Microsemi recommends selecting the UTMI interface in the USB configurator, as shown in the following figure. For interfacing the USB OTG controller with UTMI PHY, the interface signals are routed through the FPGA fabric onto the MSIOs. There is no separate I/O grouping like the ULPI interface has, for the SmartFusion2 device variants to use with UTMI PHY.

Figure 148 • MSS USB Configurator with UTMI Interface Settings



10.3.1.3 External USB PHY Reset Signal Configuration

To reset the external USB PHY, Microsemi recommends configuring the GPIO port. The following figure shows the GPIO settings in the MSS GPIO configurator in the Libero SoC design software. The GPIO port settings can be set in the I/O Editor, as shown in the following figure, to map the PIN number as per board.

Figure 149 • MSS GPIO Configurator with GPIO Settings for External USB PHY Reset

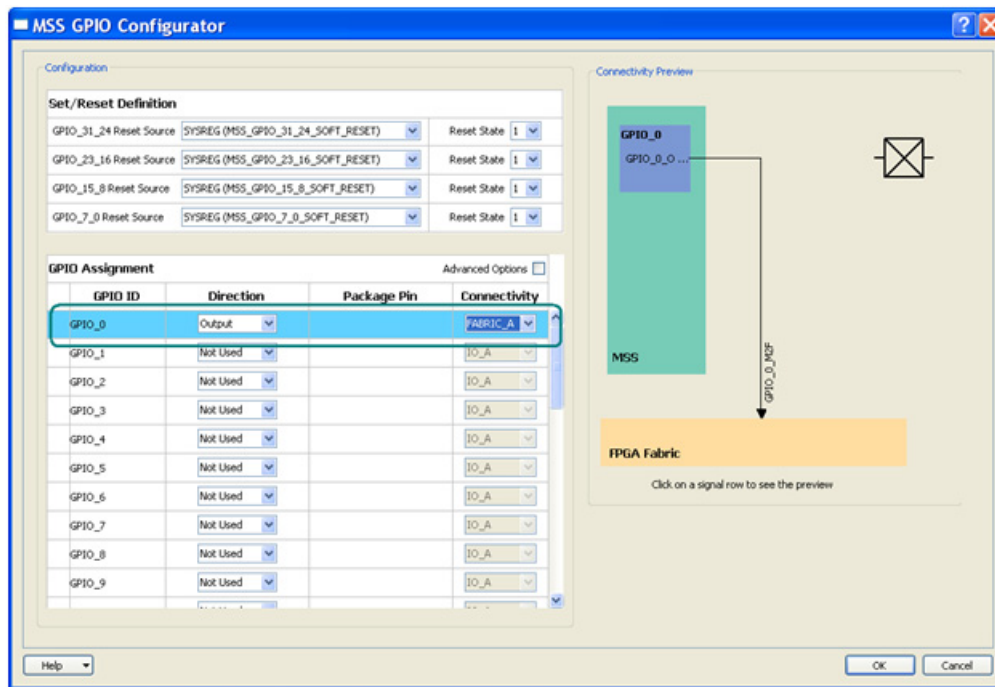
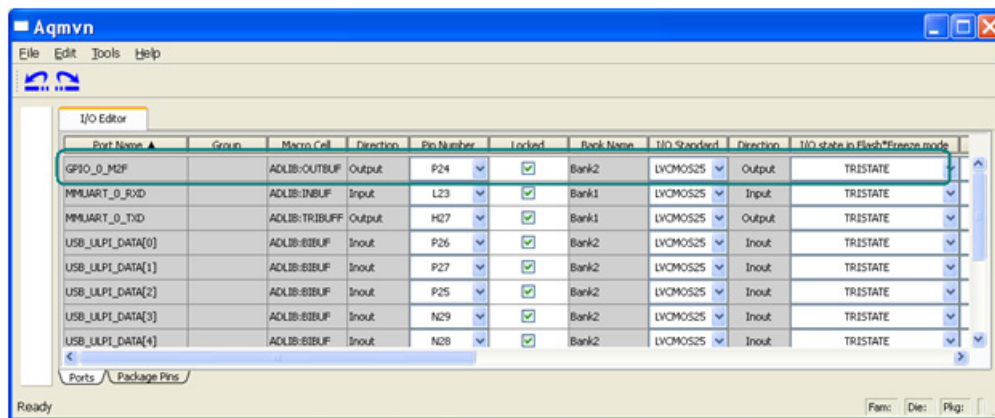


Figure 150 • I/O Editor Configurator with Settings for External USB PHY Reset Pin Mapping



All the above mentioned selections in the Libero SoC design software flow are mandatory for the applications using the USB OTG controller in SmartFusion2. Other configurations as per the complete application requirements, need to be made as well. For more details on the USB configurations using the Libero SoC design software, refer to the *Libero SoC User Guide* in the USB configurations section.

10.3.2 Software: Firmware, USB Class Specific Code, and Application Code

The embedded software flow for the USB applications vary based on the USB OTG controller role as: Host/Device. The following sections describe the embedded software flow for both the USB Device mode and the USB Host mode.

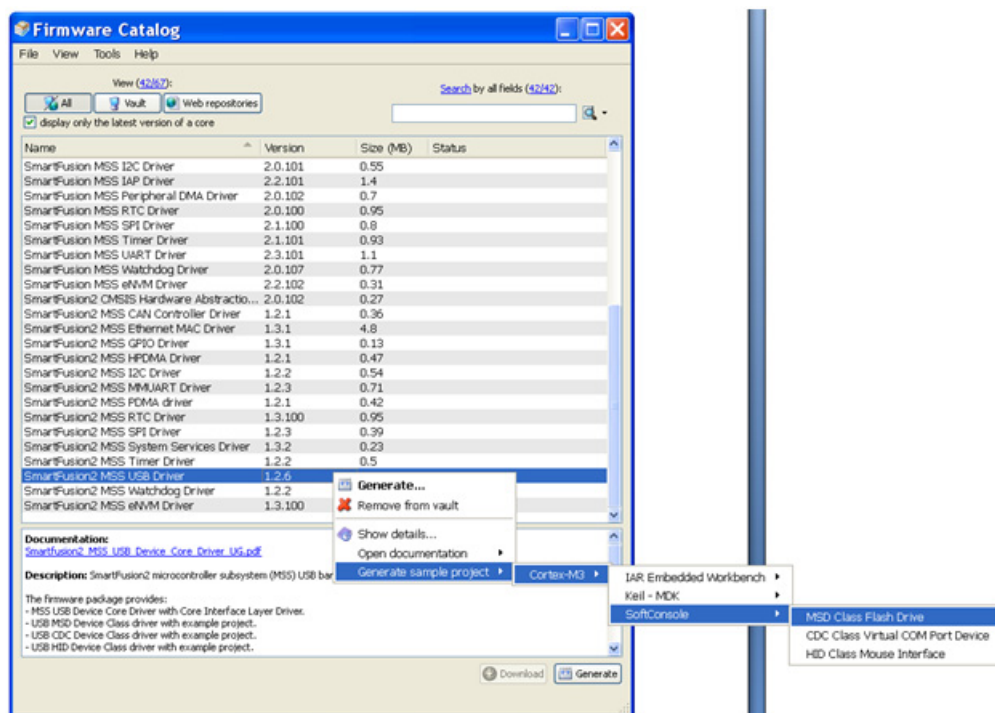
USB Device mode: As per the device mode functionality, Microsemi recommends that the USB Protocol descriptors are required to be created in the low level drivers. These descriptors are sent to the USB host which the SmartFusion2 is connected to at the time of the USB protocol enumeration processes.

If the SmartFusion2 USB OTG controller is used in USB Device mode with the USB standard class like mass storage, human interface device (HID), communications device etc. and have implemented the class driver application, then the SmartFusion2 device will be connected automatically by the USB host (for example, PC), and will perform the data transfers as per the application protocol.

If the SmartFusion2 USB OTG controller is used as a vendor-specific application with a vendor-specific-class-driver and an application code, then Microsemi recommends implementing the USB host class drivers for the USB host (for example, PC) that SmartFusion2 communicates through the USB protocol.

The firmware catalog in the Libero SoC design software, as shown in the following figure, has the reference implementations for the class drivers and application code for the USB standard mass storage, HID, and the communications device. Microsemi recommends to refer these drivers to implement the USB device application as per the requirements.

Figure 151 • Firmware Catalog with MSS USB Firmware Drivers and Sample Class Drivers



For more implementation details on the USB Protocol for the USB Device mode, refer to the **USB 2.0 specifications**.

Microsemi provides the device drivers for the USB OTG controller for SmartFusion2 and recommends using these drivers for the application development. The USB OTG drivers are implemented in two layers: USB logical driver layer (LDL) and USB core interface layer (CIL). CIL layer is the interface between the MSS USB OTG HW core and LDL.

The following table lists the APIs available in the USB firmware drivers and the description of CIL.

Table 194 • APIs available in USB Firmware Drivers and Description of CIL

API	Description
MSS_USB_core_get_configdata()	Returns the hardware (HW) configuration of the MSS USB core. This configuration cannot be changed.
MSS_USB_core_cep_flush_fifo()	Flushes the content of the Control EP FIFO.
MSS_USB_core_tx_ep_flush_fifo()	Flushes the content of transmit EP FIFO.
MSS_USB_core_rx_ep_flush_fifo()	Flushes the content of receive EP FIFO.

Table 194 • APIs available in USB Firmware Drivers and Description of CIL (continued)

API	Description
MSS_USB_core_enable_irq()	Enables USB IRQ. IRQ is selected by the parameter value provided with this function.
MSS_USB_core_disable_irq()	Disables USB IRQ. IRQ is selected by the parameter value provided with this function.
MSS_USB_core_ep_is_txfifo_notempty()	Indicates that there is at least one byte present in the Transmit endpoint FIFO
Specific to Device mode core interface layer APIs	
MSS_USB_core_get_device_info()	Returns the configuration of the MSS USB core. This configuration cannot be changed.
MSS_USB_core_device_set_address()	Sets the address of the MSS USB in Device mode. This value generally is assigned by the USB host to identify the device.
MSS_USB_core_device_get_address()	Returns the address set for the MSS USB in the Device mode. This value generally is assigned by the USB host to identify the device.
MSS_USB_core_device_force_resume()	Forces the MSS USB core to exit Suspend mode and generate resume signaling when operating in Device mode.
MSS_USB_core_device_set_isoupdate()	Forces ISO endpoint from the time data is deemed ready to wait for an SOF token before IN token is received. If IN token is received before an SOF token, then a zero length data packet will be sent.
MSS_USB_core_device_clr_isoupdate()	Configures the ISO EP so that, once the data is deemed ready, ISO endpoint will not wait for SOF token to arrive before IN token. Data will be transmitted on the next received IN token.

Table 195 • Device Mode Class Driver APIs (LDL)

API	Description
MSS_USB_device_set_desc_cb_handler()	Provides the call-back functions for USB descriptors to this driver.
MSS_USB_device_set_class_cb_handler()	Provides the call-back functions for USB class implementation to this driver.
MSS_USB_device_init()	Initializes the MSS USB to operate in Device mode at desired USB speed.
MSS_USB_device_cep_configure()	Configures the Control EP. Parameters for Control EP such as max_pkt_size, FIFO address, and FIFO size in the MSS core are fixed.
MSS_USB_device_cep_read_prepare()	Prepares the Control EP for receiving control transfer packet.
MSS_USB_device_cep_write()	Transmits data on Control EP.
MSS_USB_device_cep_error()	Stalls the CEP when there is a error in data exchange or when the USB request is not supported.
MSS_USB_device_cep_clr_error()	Clears the stall condition on previously stalled control endpoint.
MSS_USB_device_tx_ep_configure()	Configures the transmit EP (USB IN Transfers) as per the provided parameters.
MSS_USB_device_rx_ep_configure()	Configures the receive EP (USB OUT Transfers) as per the provided parameters.
MSS_USB_device_rx_ep_read_prepare()	Prepares receive EP for receiving data packets. (USB OUT Transfers).
MSS_USB_device_tx_ep_write()	Transmits data on transmit endpoint (USB IN Transfers).
MSS_USB_device_tx_ep_error()	Stalls transmit endpoint when there is error in data exchange.
MSS_USB_device_rx_ep_error()	Stalls receive endpoint when there is error in data exchange.
MSS_USB_device_tx_ep_clr_error()	Clears the stall condition on previously stalled transmit endpoint.
MSS_USB_device_rx_ep_clr_error()	Clears the stall condition on previously stalled receive endpoint.

The USB OTG controller device driver framework is made flexible to the application/class specific driver implementations by providing the callback functions. These call back functions are to be implemented by the class driver which will have class specific commands and responses.

The following table shows the call back function prototype and refer to the example projects provided to know how to use these call back functions in more details.

Table 196 • Functional Descriptions of Callback APIs

Callback API	Description
uint8_t (*usb_class_init) (uint8_t cfgidx);	This function is called when USB device receives SET_CONFIGURATION request from USB host with a non-zero cfgidx number. The parameter cfgidx indicates the configuration number that should be set by the USB device.
uint8_t (*usb_class_deinit) (uint8_t cfgidx);	This function is called when USB device receives SET_CONFIGURATION request from the USB host with a cfgidx = 0. The parameter cfgidx indicates the configuration number that should be set by the USB device. USB device goes into the MSS_USB_ADDRESS_STATE on receiving this command.

Table 196 • Functional Descriptions of Callback APIs (continued)

Callback API	Description
uint8_t* (*usb_class_get_descriptor) (uint8_t recipient, uint8_t type, uint32_t* length);	This function is called when USB device receives GET_DESCRIPTOR request from USB host requesting a class specific descriptor (configuration, class, interface, endpoint, OTG, vendor descriptors). The parameter recipient indicates the intended recipient by the USB host (endpoint, interface or device). The parameter type indicates the type of descriptor requested.
uint8_t (*usb_class_process_requests) (uint8_t** buf_p, uint8_t request, uint32_t* length, uint8_t* data_buf_p, uint32_t data_len);	This function is called when USB device receives class specific request from USB host. The parameter request indicates the class specific request that need to be processed. The function should return a pointer to the data requested structure in the return parameter buf_p and length of the buffer in the return parameter length. If the host provides data along with the request to be processed, a pointer to this data will be passed in the parameter data_buf_p, and the size of the buffer will be passed in the parameter data_len.
uint8_t (*usb_class_datain) (mss_usb_ep_num_t num, uint8_t status);	This function is called when provided data is transferred on previously configured transmit endpoint. The endpoint on which data is received is indicated by parameter num. The parameter status indicates error status of the transmit transaction. A non-zero status value indicates that there was error in last receive transaction.
uint8_t (*usb_class_dataout) (mss_usb_ep_num_t num, uint8_t status, uint32_t rx_count);	This function is called when data is received on previously configured receive endpoint. The endpoint on which data is received is indicated by parameter num. The parameter status indicates error status of the receive transaction. A non-zero status value indicates that there was error in last receive transaction. The rx_count parameter indicates the number of bytes received in the last receive transaction.
uint8_t (*usb_class_cep_datain) (uint8_t status);	This function is called when data packet is transmitted on previously configured control endpoint. The parameter status indicates error status of the transmit transaction. A non-zero status value indicates that there was error in last transmit transaction.
uint8_t (*usb_class_cep_dataout) (uint8_t status);	This function is called when data packet is received on previously configured control endpoint. The parameter status indicates error status of the receive transaction. A non-zero status value indicates that there was error in last receive transaction.
uint8_t (*usb_device_descriptor) (uint32_t* length);	This function is called when USB device receives the GET_DESCRIPTOR command requesting device descriptor from USB host. This function should return a pointer to the device descriptor and provide the length of the descriptor in the return parameter.
uint8_t (*usb_device_qual_descriptor) (mss_usb_device_speed_t speed, uint32_t* length);	This function is called when USB device receives the GET_DESCRIPTOR command requesting device qualifier descriptor from USB host. This function should return a pointer to the device qualifier descriptor and provide the length of the descriptor in the return parameter.
uint8_t (*usb_string_descriptor) (uint8_t index, uint32_t* length);	This function is called when USB device receives the GET_DESCRIPTOR command requesting specific string descriptor from USB host. Requested string descriptor number is provided in parameter index. This function should return a pointer to the requested string descriptor and provide the length of the descriptor in the return parameter length.

Refer to the **SmartFusion2 MSS USB Device Core Driver's User Guide** for more information on the API detailed description and parameters to the APIs.

10.3.3 USB OTG Controller Clocks and Resets

The following conditions are to be considered in the design for clocks and resets while using the USB OTG controller:

- In order to operate the USB OTG controller correctly with the external USB PHY device, the MSS FCLK and AHB CLK must be configured to run at greater than 30 MHz.
- In order to reset the external USB PHY, it is necessary to use an MSS GPIO port. Firmware resets the USB PHY whenever there is a USB controller reset done by the USB soft reset register settings.
- The USB controller resets on power-up and is held in reset until it is enabled. The USB controller can be reset by writing to USB_SOFTRESET field of the SOFTRESET_REG at address 0x40038048, located in the SYSREG block. The USB firmware drivers implements this feature.

Table 197 • SOFTRESET_REG Bit for USB Controller Soft Reset

Register Name	Address	Bit Number	Name	Reset Value	Function
SOFT_RESET_REG (0x4004307F)	0x40038048	14	USB_SOFTRESET	0x1	Controls reset input to the USB controller 0: Release USB controller from reset 1: Keep USB controller in reset (reset value)

At power-up this bit is asserted as 1. This keeps the USB controller in a reset state. If the bit is set to 0, the USB controller is allowed to become active. If USB_SOFTRESET is 0, the USB controller could still be held in reset by other system reset sources.

10.3.4 Programmability

10.3.4.1 Memory Map

The address space of the USB OTG controller in SmartFusion2 is from 0x40043000 to 0x40043FFF. All the USB OTG FIFO registers are residing in this address space.

10.3.4.2 USB OTG Controller Registers Map

This section describes the register map; bit description of various categories of registers in the USB controller. In addition, System Registers that are applicable to USB are described in this section. This provides programmers information for firmware development. Microsemi recommends using the drivers provided in the tool set for application development.

The Register set in the USB controller consists of the following categories:

- Common Registers: Provide control and status for the USB controller.
- Indexed Registers: Provide control and status for the currently selected end point (host or peripheral).
- FIFO Registers: Provide access to end point FIFOs in the USB controller.
- Control and Configuration Registers: Provide additional device status and control.
- Non-Indexed End Point Control/Status Registers: Are accessible independently for every endpoint, whereas indexed registers are shared by endpoints. These cover EP0, EP1, EP2, EP3, and EP4.
- Extended Registers: Provides details on additional registers that control and affect the operation of the USB controller.
- DMA Registers: Provide control and status of built-in DMA.
- Multipoint Control and Status Registers: Details additional control and status registers that relate to the multipoint option. These registers are required and have relevance in Host mode only.
- LPM Registers: These correspond to LPM.
- USB System Registers: Are SmartFusion2 system registers that are associated with USB.

10.3.5 Common Registers

This section covers all registers in this category along with the address offset, functionality, and per bit details.

Table 198 • Common Register Set Description

Register Name	Address	Width	R/W Type	Reset Value	Description
FADDR_REG(0x40043000)	0x40043000	8	RW	0	Write with the 7-bit address of the peripheral part of the transaction. This register applies to operations when the USB controller is used in peripheral mode only. It is ignored in Host mode.
POWER_REG (0x40043001)	0x40043001	8	R	0x20	Controls suspend and resume signaling and some other basic operational aspects of the USB controller.
TX_IRQ_REG (0x40043002)	0x40043002	16	R	0	Indicates which interrupts are active for endpoint 0 and transmit endpoints EP1, EP2, EP3, and EP4. These are the lowest 5 bits of the register. Interrupts are cleared when this register is read.
RX_IRQ_REG (0x40043004)	0x40043004	16	R	0	Indicates which interrupts are active for receive endpoints EP1, EP2, EP3, and EP4. These are bits 1, 2, 3, and 4 of the register. Interrupts are cleared when this register is read.
TX_IRQ_EN_REG (0x40043006)	0x40043006	16	RW	0x1F	Provides interrupt enables for interrupts in TX_IRQ_REG. The endpoint0 and EP1, EP2, EP3, and EP4 have corresponding enable bits from bit 0 to bit 4 of this register. A value of 1 indicates the interrupt is enabled.
RX_IRQ_EN_REG (0x40043008)	0x40043008	16		0x1E	Provides interrupt enables for interrupts in RX_IRQ_REG. The endpoints EP1, EP2, EP3, and EP4 have corresponding enable bits from bit 1 to bit 4 of this register. A value of 1 indicates the interrupt is enabled.
USB_IRQ_REG (0x4004300A)	0x4004300A	8	R	0	Indicates the status of USB interrupts. All active interrupts are cleared when the register is read.
USB_IRQ_EN_REG (0x4004300B)	0x4004300B	8	RW	0x06	Provides interrupt enables for interrupts in USB_IRQ_REG. A value of 1 indicates the interrupt is enabled.
FRAME_REG (0x4004300C)	0x4004300C	16	R	9	Holds the last received frame number. This is an 11-bit number.
INDEX_REG (0x4004300E)	0x4004300E	4	RW	0	Indicates which endpoint control and status registers are currently accessed from among the implemented transmit and receive endpoints (EP0, EP1, EP2, EP3, and EP4). Each transmit endpoint and each receive endpoint has its own set of control/status registers located between 0x40043100 and 0x400431FF addresses. In addition, one set of TX control/status and one set of RX control/status registers appear at 10h to 19h. Before accessing an endpoint's control/status registers at 10h to 19h, the endpoint number should be written to the Index register to ensure that the correct control/status registers appear in the memory map.

Table 198 • Common Register Set Description (continued)

Register Name	Address	Width	R/W Type	Reset Value	Description
TEST_MODE_REG (0x4004300F)	0x4004300F	8	RW	0	Puts the USB controller in one of the four test modes for high speed operation described in the USB 2.0 specification. This register is not used in normal operation.

10.3.5.1 FADDR_REG Bit Definitions

Table 199 • FADDR_REG(0x40043000)

Bit Number	Name	Reset Value	Function
7	Reserved	0	N/A
[6:0]	Func Addr	0	Function Address: Write with the 7-bit address of the peripheral part of the transaction. This register applies to operations when the USB controller is used in Peripheral mode only. It is ignored in Host mode.

10.3.5.2 POWER_REG Bit Definitions

Table 200 • POWER_REG (0x40043001)

Bit Number	Name	Reset Value	Function
7	ISO Update	0	When set by the Cortex-M3 processor (or fabric master), the USB controller waits for an SOF token from the time TxPktRdy is set before sending the packet. If an IN token is received before an SOF token, a zero length data packet will be sent. Only valid in Peripheral mode. Also, this bit only affects endpoints performing ISO transfers.
6	Soft Conn	0	If the soft connect/disconnect feature is enabled, the USB D+/D- lines are enabled when this bit is set by the Cortex-M3 processor (or fabric master) and tristated when this bit is cleared by the Cortex-M3 processor (or fabric master). Only valid in Peripheral mode.
5	HS Enab	1	When set by the CPU, the USB controller negotiates for High speed mode when the device is reset by the hub. If not set, the device will only operate in Full speed mode.
4	HS Mode	0	When set, this read-only bit indicates High speed mode successfully negotiated during USB reset. In Peripheral mode, becomes valid when USB reset completes (as indicated by USB reset interrupt). In Host mode, becomes valid when the reset bit is cleared. Remains valid for the duration of the session. Allowance is made for Tiny-J signaling in determining the transfer speed to select.
3	Reset	0	This bit is set when reset signaling is present on the bus. This bit is read/write from the Cortex-M3 processor (or fabric master) in Host mode but read-only in Peripheral mode.

Table 200 • POWER_REG (0x40043001) (continued)

Bit Number	Name	Reset Value	Function
2	Resume	0	Set by the Cortex-M3 processor (or fabric master) to generate resume signaling when the device is in Suspend mode. In Peripheral mode, the Cortex-M3 processor (or fabric master) should clear this bit after 10 ms (a maximum of 15 ms), to end resume signaling. In Host mode, the Cortex-M3 processor (or fabric master) should clear this bit after 20 ms.
1	Suspend Mode	0	In Host mode, this bit is set by the Cortex-M3 processor (or fabric master) to enter Suspend mode. In Peripheral mode, this bit is set on entry into Suspend mode. It is cleared when the Cortex-M3 processor (or fabric master) reads the interrupt register, or sets the resume bit above.
0	Enable SuspendM	0	Set by the Cortex-M3 processor (or fabric master) to enable the SUSPENDM output.

10.3.5.3 TX_IRQ_REG Bit Definitions

Table 201 • TX_IRQ_REG (0x40043002)

Bit Number	Name	Reset Value	Function
[15:5]	Reserved	N/A	N/A
4	EP4 Tx	0	Transmit endpoint 4 interrupt
3	EP3 Tx	0	Transmit endpoint 3 interrupt
2	EP2 Tx	0	Transmit endpoint 2 interrupt
1	EP1 Tx	0	Transmit endpoint 1 interrupt
0	EP0	0	Endpoint 0 interrupt

10.3.5.4 RX_IRQ_REG Bit Definitions

Table 202 • RX_IRQ_REG (0x40043004)

Bit Number	Name	Reset Value	Function
[15:5]	Reserved	N/A	N/A
4	EP4 Rx	0	Receive endpoint 4 interrupt
3	EP3 Rx	0	Receive endpoint 3 interrupt
2	EP2 Rx	0	Receive endpoint 2 interrupt
1	EP1 Rx	0	Receive endpoint 1 interrupt
0	Reserved	0	Always returns zero

10.3.5.5 TX_IRQ_EN_REG Bit Definitions

Table 203 • TX_IRQ_EN_REG (0x40043006)

Bit Number	Name	Reset Value	Function
[15:5]	Reserved	N/A	N/A
4	EP4 TxEn	1	Transmit endpoint 4 interrupt enable
3	EP3 TxEn	1	Transmit endpoint 3 interrupt enable
2	EP2 TxEn	1	Transmit endpoint 2 interrupt enable
1	EP1 TxEn	1	Transmit endpoint 1 interrupt enable
0	EP0	1	Endpoint0 interrupt enable

10.3.5.6 RX_IRQ_EN_REG Bit Definitions

Table 204 • RX_IRQ_EN_REG (0x40043008)

Bit Number	Name	Reset Value	Function
[15:5]	Reserved	N/A	N/A
4	EP4 RxEn	1	Receive endpoint 4 interrupt enable
3	EP3 RxEn	1	Receive endpoint 3 interrupt enable
2	EP2 RxEn	1	Receive endpoint 2 interrupt enable
1	EP1 RxEn	1	Receive endpoint 1 interrupt enable
0	Reserved	0	Always returns zero

10.3.5.7 USB_IRQ_REG Bit Definitions

Table 205 • USB_IRQ_REG (0x4004300A)

Bit Number	Name	Reset Value	Function
7	VBus Error	0	Set when VBus drops below the VBus valid threshold during a session. Only valid when the USB controller is an A device.
6	Sess Req	0	Set when session request signaling has been detected. Only valid when the USB controller is an A device.
5	Discon	0	Set in Host mode when a device disconnection is detected. Set in Peripheral mode when a session ends. Valid at all transaction speeds.
4	Conn	0	Set when a device connection is detected. Only valid in Host mode. Valid at all transaction speeds.
3	SOF	0	Set when a new frame starts.
2	Reset	0	Set in Peripheral mode when reset signaling is detected on the bus.
	Babble		Set in Host mode when babble is detected. Only active after first SOF has been sent.
1	Resume	0	Set when resume signaling is detected on the bus while the USB controller is in Suspend mode.

Table 205 • USB_IRQ_REG (0x4004300A)

Bit Number	Name	Reset Value	Function
0	Suspend	0	Set when suspend signaling is detected on the bus. Only valid in Peripheral mode.

10.3.5.8 USB_IRQ_EN_REG Bit Definitions

Table 206 • USB_IRQ_EN_REG (0x4004300B)

Bit Number	Name	Reset Value	Function
7	VbEn	0	Interrupt Enable for corresponding bit[7], VBus Error in USB_IRQ_REG
6	SrEn	0	Interrupt Enable for corresponding bit[6], Sess Req in USB_IRQ_REG
5	DcEn	0	Interrupt Enable for corresponding bit[5], Discon in USB_IRQ_REG
4	CoEn	0	Interrupt Enable for corresponding bit[4], Conn in USB_IRQ_REG
3	SofEn	0	Interrupt Enable for corresponding bit[3], SOF in USB_IRQ_REG
2	ReBaEn	1	Interrupt Enable for corresponding bit[2], Reset/Babble in USB_IRQ_REG
1	ReEn	1	Interrupt Enable for corresponding bit[1], Resume in USB_IRQ_REG
0	SuEn	0	Interrupt Enable for corresponding bit[0], Suspend in USB_IRQ_REG

10.3.5.9 FRAME_REG Bit Definitions

Table 207 • FRAME_REG (0x4004300C)

Bit Number	Name	Reset Value	Function
[15:11]	Reserved	N/A	N/A
[10:0]	Frame Number	0	This is the 11-bit frame number, with bit[10] being MSB and bit[0] being LSB.

10.3.5.10 INDEX_REG Bit Definitions

Table 208 • INDEX_REG (0x4004300E)

Bit Number	Name	Reset Value	Function
[3:0]	Selected Endpoint	0	This is the index into the selected endpoint.

10.3.5.11 TEST_MODE_REG Bit Definitions

Table 209 • TEST_MODE_REG (0x4004300F)

Bit Number	Name	Reset Value	Function
7	Force_Host	0	The Cortex-M3 processor (or fabric master) sets this bit to instruct the core to enter Host mode when the session bit is set, regardless of whether it is connected to any peripheral. The state of the CID input, HostDisconnect, and LineState signals is ignored. The core will then remain in Host mode until the session bit is cleared, even if a device is disconnected, and if the Force_Host bit remains set, will re-enter Host mode the next time the session bit is set. While in this mode, the status of the HOSTDISCON signal from the PHY may be read from bit 7 of the DevCtl register. The operating speed is determined from the Force_HS and Force_FS bits, as shown in Table 210 , page 314.
6	FIFO_Access	0	The Cortex-M3 processor (or fabric master) sets this bit to transfer the packet in the endpoint 0 TX FIFO to the endpoint 0 Rx FIFO. This bit is cleared automatically.
5	Force_FS	0	The Cortex-M3 processor (or fabric master) sets this bit either in conjunction with bit 7 above or to force the USB controller into Full speed mode when it receives a USB reset.
4	Force_HS	0	The Cortex-M3 processor (or fabric master) sets this bit either in conjunction with bit 7 above or to force the USB controller into High speed mode when it receives a USB reset.
3	Test_Packet	0	(High speed mode) The Cortex-M3 processor (or fabric master) sets this bit to enter Test_Packet test mode. In this mode, the USB controller repetitively transmits on the bus a 53-byte test packet, the form of which is defined in the USB specification revision 2.0, Section 7.1.20 (and in section 28.4). The test packet has a fixed format and must be loaded into the endpoint 0 FIFO before the test mode is entered.
2	Test_K	0	(High speed mode) The Cortex-M3 processor (or fabric master) sets this bit to enter Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1	Test_J	0	(High speed mode) The Cortex-M3 processor (or fabric master) sets this bit to enter Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.
0	Test_SE0_NAK	0	(High speed mode) The Cortex-M3 processor (or fabric master) sets this bit to enter Test_SE0_NAK test mode. In this mode, the USB controller remains in high speed mode but responds to any valid IN token with a NAK.

10.3.5.12 Operating Speed

Table 210 • Operating Speed

Bit Number	Reset Value	Function
0	0	Low speed
0	1	Full speed
1	0	High speed
1	1	Undefined

10.3.6 Indexed Registers

This section covers all the registers in this category along with the address offset, functionality, and per-bit details. The registers mapped into this section depend on whether the core is in Peripheral mode (DEV_CTRL_REG.Bit2 = 0) or in Host mode (DEV_CTRL_REG.Bit2 = 1) and the value of the Index register (INDEX_REG).

Table 211 • Indexed Register Set Description

Register Name	Address	Width	R/W Type	Reset Value	Description
TX_MAX_P_REG	0x40043010	16	RW	0	Maximum packet size of host transmit endpoint. (Index register set to select endpoints 1 – 4 only).
CSR0L_REG (Peripheral) CSR0L_REG (Host) TX_CSRL_REG (Peripheral) TX_CSRL_REG (Host)	0x40043012	8	RW	0	Provide control and status bits for endpoint0 (Index register set to select endpoint0) and transmit endpoint0 (Index register set to select endpoints 1 – 4). The interpretation of the register depends on whether the USB controller is acting as a peripheral or a host. The value returned when the register is read reflects the status attained; for example, as a result of writing to the register.
CSR0H_REG (Peripheral) CSR0H_REG (Host) TX_CSRH_REG (Peripheral) TX_CSRH_REG (Host)	0x40043013	8	RW	0	Provides control and status bits for endpoint0 (Index register set to select endpoint0) and transmit endpoint0 (Index register set to select endpoints 1 – 4). The interpretation of the register depends on whether the USB controller is acting as a peripheral or a host. The value returned when the register is read reflects the status attained; for example, as a result of writing to the register.
RX_MAX_P_REG	0x40043014	16	RW	0	Defines the maximum amount of data that can be transferred through the selected receive endpoint in a single operation. There is one such register for each receive endpoint (except endpoint 0).
RX_CSRL_REG (Peripheral) RX_CSRL_REG (Host)	0x40043016	8	RW	0	Provide control and status bits for transfers through the currently selected receive endpoint. There is one such register for each configured receive endpoint (not including endpoint 0). The interpretation of the register depends on whether the USB controller is acting as a peripheral or a host. The value returned when the register is read reflects the status attained; for example, as a result of writing to the register.
RX_CSRH_REG (Peripheral) RX_CSRH_REG (Host)	0x40043017	8	RW	0	Provides control and status bits for transfers through the currently-selected receive endpoint (Index register set to select endpoints 1 – 4 only). There is one such register for each configured receive endpoint (not including endpoint 0). The interpretation of the register depends on whether the USB controller is acting as a peripheral or a host. The value returned when the register is read reflects the status attained; for example, as a result of writing to the register.

Table 211 • Indexed Register Set Description (continued)

Register Name	Address	Width	R/W Type	Reset Value	Description
COUNT0_REG	0x40043018	7	R	0	Indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while RxPktRdy (CSR0L_REG.bit0) is set.
RX_COUNT_REG	0x40043018	14	R	0	Holds the number of data bytes in the packet currently in line to be read from the Rx FIFO. If the packet is transmitted as multiple bulk packets, the number given will be for the combined packet. The value returned changes as the FIFO is unloaded and is only valid while RxPktRdy (CSR0L_REG.bit0) is set.
TYPE0_REG	0x4004301A	8	RW	0	Applicable in Host mode only. Number of received bytes in endpoint0 FIFO (Index register set to select endpoint 0).
TX_TYPE_REG	0x4004301A	8	RW	0	Number of bytes to be read from the peripheral receive endpoint FIFO (Index register set to select endpoints 1 – 4).
NAK_LIMIT0_REG	0x4004301B	5	RW	0	Sets the NAK response timeout on endpoint 0 (Index register set to select endpoint 0).
TX_INTERVAL_REG	0x4004301B	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host transmit endpoint (Index register set to select endpoints 1 – 4 only).
RX_TYPE_REG	0x4004301C	8	RW	0	Sets the transaction protocol, speed, and peripheral endpoint number for the host receive endpoint (Index register set to select endpoints 1 – 4 only).
RX_INTERVAL_REG	0x4004301D	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host receive endpoint (Index register set to select endpoints 1 – 4 only).
CONFIG_DATA_REG	0x4004301F	8	R		Returns details of USB controller configuration (Index register set to select endpoint 0).
FIFO_SIZE_REG	0x4004301F	8	R		Returns the configured size of the selected receive and transmit FIFOs (endpoints 1 – 4 only).

10.3.6.1 TX_MAX_P_REG Bit Definitions

Table 212 • TX_MAX_P_REG

Bit Number	Name	Reset Value	Function
[15:11]	m-1	N/A	<p>If the core is configured with high-bandwidth ISO/interrupt endpoints or packet splitting on bulk endpoints, the register includes 2 or 5 further bits that define a multiplier <i>m</i> which is equal to one more than the value recorded.</p> <p>For bulk endpoints with the packet splitting option enabled, the multiplier <i>m</i> can be up to 32 and defines the maximum number of USB packets (packets for transmission over the USB) of the specified payload into which a single data packet placed in the FIFO should be split, prior to transfer. If the packet splitting option is not enabled, bits[15:13] are not implemented and bits[12:11] (if included) are ignored.</p> <p>The data packet must be an exact multiple of the payload specified by bits 10:0, which is itself required to be either 8, 16, 32, 64, or (in the case of high speed transfers) 512 bytes.</p> <p>For ISO/interrupts endpoints operating in High speed mode and with the high-bandwidth option enabled, <i>m</i> may only be 2 or 3 (corresponding to bit[11] set or bit[12] set) and it specifies the maximum number of such transactions that can take place in a single microframe.</p> <p>If either bit 11 or bit 12 is non-zero, the USB controller automatically splits any data packet written to the FIFO into 2 or 3 USB packets, each containing the specified payload (or less). The maximum payload for each transaction is 1,024 bytes, so this allows up to 3,072 bytes to be transmitted in each microframe. For ISO transfers in Full speed mode or if high-bandwidth is not enabled, bits[11] and [12] are ignored.</p> <p>The value written to bits[10:0] (multiplied by <i>m</i> in the case of high-bandwidth ISO transfers) must match the value given in the <i>wMaxPacketSize</i> field of the standard endpoint descriptor for the associated endpoint (refer to the <i>USB specification v2.0</i>, Chapter 9). A mismatch could cause unexpected results.</p> <p>The total amount of data represented by the value written to this register (specified payload × <i>m</i>) must not exceed the FIFO size for the transmit endpoint, and should not exceed half the FIFO size if double-buffering is required.</p> <p>If this register is changed after packets have been sent from the endpoint, the transmit endpoint FIFO should be completely flushed (using the FlushFIFO bit, TX_CSRL_REG.bit[3]) after writing the new value to this register.</p>
[10:0]	TxMaxP	m-1	<p>Maximum payload/transaction.</p> <p>Maximum payload in bytes transmitted in a single transaction. The value set can be up to 1,024 bytes but is subject to the constraints placed by the USB specification on packet sizes for bulk, interrupt, and ISO transfers in full speed and high speed operations. This must be set to an even number of bytes for proper interrupt generation DMA mode 1.</p>

10.3.6.2 CSROL_REG (in Peripheral mode) Bit Definitions

Table 213 • CSROL_REG (Peripheral)

Bit Number	Name	Reset Value	Function
7	ServicedSetupEnd	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to clear the SetupEnd bit (bit[4] of this register). This bit is self clearing.
6	ServicedRxPktRdy	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to clear the RxPktRdy bit (bit[0] of this register). This bit is self clearing.

Table 213 • CSROL_REG (Peripheral) (continued)

Bit Number	Name	Reset Value	Function
5	SendStall	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to terminate the current transaction. The STALL handshake will be transmitted and then this bit will be cleared automatically.
4	SetupEnd	0	This bit is set when a control transaction ends before the DataEnd bit (bit[3] of this register) has been set. An interrupt is generated and the FIFO is flushed at this time. The bit is cleared by the Cortex-M3 processor (or fabric master) writing a 1 to the ServicedSetupEnd bit (bit[7] of this register).
3	DataEnd	0	The Cortex-M3 processor (or fabric master) sets this bit: – When setting TxPktRdy (bit[1] of this register) for the last data packet. – When clearing RxPktRdy (bit[0] of this register) after unloading the last data packet. – When setting TxPktRdy (bit[1] of this register) for a zero length data packet. It is cleared automatically.
2	SentStall	0	This bit is set when a STALL handshake is transmitted. The Cortex-M3 processor (or fabric master) should clear this bit.
1	TxPktRdy	0	The Cortex-M3 processor (or fabric master) sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled).
0	RxPktRdy	0	This bit is set when a data packet has been received. An interrupt is generated when this bit is set. The Cortex-M3 processor (or fabric master) clears this bit by setting the ServicedRxPktRdy bit (bit[6] of this register).

10.3.6.3 CSROL_REG (Host mode) Bit Definitions

Table 214 • CSROL_REG (Host)

Bit Number	Name	Reset Value	Function
7	NAK Timeout	0	This bit is set when endpoint 0 is halted, following the receipt of NAK responses for longer than the time set by the NAKLimit0 register. The Cortex-M3 processor (or fabric master) should clear this bit to allow the endpoint to continue.
6	StatusPkt	0	The Cortex-M3 processor (or fabric master) sets this bit at the same time as the TxPktRdy (bit[1] of this register) or ReqPkt bit (bit 0 of this register) is set, to perform a status stage transaction. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.
5	ReqPkt	0	The Cortex-M3 processor (or fabric master) sets this bit to request an IN transaction. It is cleared when RxPktRdy (bit [0] of this register) is set.
4	Error	0	This bit will be set when three attempts have been made to perform a transaction with no response from the peripheral. The Cortex-M3 processor (or fabric master) should clear this bit. An interrupt is generated when this bit is set.
3	SetupPkt	0	The Cortex-M3 processor (or fabric master) sets this bit, at the same time as the TxPktRdy bit (bit[1] of this register) is set, to send a SETUP token instead of an OUT token for the transaction. Setting this bit also clears the Data Toggle.
2	RxStall	0	This bit is set when a STALL handshake is received. The Cortex-M3 processor (or fabric master) should clear this bit.

Table 214 • CSR0L_REG (Host) (continued)

Bit Number	Name	Reset Value	Function
1	TxPktRdy	0	The Cortex-M3 processor (or fabric master) sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled).
0	RxPktRdy	0	This bit is set when a data packet has been received. An interrupt is generated (if enabled) when this bit is set. The Cortex-M3 processor (or fabric master) should clear this bit when the packet has been read from the FIFO.

10.3.6.4 CSROH_REG (in Peripheral mode) Bit Definitions

Table 215 • CSROH_REG (Peripheral)

Bit Number	Name	Reset Value	Function
[7:1]	Reserved	N/A	
0	FlushFIFO	0	The Cortex-M3 processor (fabric master) writes a 1 to this bit to flush the next packet to be transmitted/read from the endpoint 0 FIFO. The FIFO pointer is reset and the TxPktRdy/RxPktRdy bit (bit[1] and bit[0] of CSR0L_REG) is cleared. FlushFIFO should only be used when TxPktRdy/RxPktRdy is set. At other times, it may cause data to be corrupted.

10.3.6.5 CSROH_REG (in Host mode) Bit Definitions

Table 216 • CSROH_REG (Host)

Bit Number	Name	Reset Value	Function
[7:4]	Reserved	N/A	
3	Dis Ping	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to instruct the USB controller not to issue PING tokens in data and status phases of a high speed control transfer (for use with devices that do not respond to PING).
2	Data Toggle Write Enable	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to enable the current state of the endpoint 0 data toggle to be written (refer to the Data Toggle bit). This bit is automatically cleared once the new value is written.
1	Data Toggle	0	When read, this bit indicates the current state of the endpoint 0 data toggle. If Data Toggle Write Enable (bit 2 of this register) is High, this bit may be written with the required setting of the data toggle. If Data Toggle Write Enable is Low, any value written to this bit is ignored.
0	FlushFIFO	0	The Cortex-M3 processor (fabric master) writes a 1 to this bit to flush the next packet to be transmitted/read from the endpoint 0 FIFO. The FIFO pointer is reset and the TxPktRdy/RxPktRdy bit (bit 1 and bit 0 of CSR0L_REG) is cleared. FlushFIFO should only be used when TxPktRdy/RxPktRdy is set. At other times, it may cause data to be corrupted.

10.3.6.6 TX_CSRL_REG (in Peripheral mode) Bit Definitions

Table 217 • TX_CSRL_REG (Peripheral)

Bit Number	Name	Reset Value	Function
7	IncompTx	0	When the endpoint is being used for high-bandwidth ISO transfers, this bit is set to indicate where a large packet has been split into 2 or 3 packets for transmission but insufficient IN tokens have been received to send all the parts. In anything other than ISO transfers, this bit will always return 0.
6	ClrDataTog	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to reset the endpoint data toggle to 0.
5	SentStall	0	This bit is set when a STALL handshake is transmitted. The FIFO is flushed and the TxPktRdy bit (bit 0 of this register) is cleared. The Cortex-M3 processor (or fabric master) should clear this bit.
4	SendStall	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to issue a STALL handshake to an IN token. The Cortex-M3 processor (or fabric master) clears this bit to terminate the stall condition. This bit has no effect where the endpoint is being used for ISO transfers.
3	FlushFIFO	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to flush the latest packet from the endpoint TX FIFO. The FIFO pointer is reset, the TxPktRdy bit (bit0 of this register) is cleared and an interrupt is generated. It may be set simultaneously with TxPktRdy to abort the packet that is currently being loaded into the FIFO. FlushFIFO should only be used when TxPktRdy is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
2	UnderRun	0	The controller sets this bit if an IN token is received when TxPktRdy (bit0 of this register) is not set. The Cortex-M3 processor (or fabric master) should clear this bit.
1	FIFONotEmpty	0	The controller sets this bit when there is at least 1 packet in the transmit FIFO.
0	TxPktRdy	0	The Cortex-M3 processor (or fabric master) sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is generated at this point (if enabled). TxPktRdy is automatically cleared prior to loading a second packet into a double-buffered FIFO.

10.3.6.7 TX_CSRL_REG (in Host mode) Bit Definitions

Table 218 • TX_CSRL_REG (Host)

Bit Number	Name	Reset Value	Function
7	NAK Timeout	0	<i>(Bulk endpoints only)</i> This bit will be set when the transmit endpoint is halted, following the receipt of responses for longer than the time set as the NAK Limit by the TxInterval register. The Cortex-M3 processor (or fabric master) should clear this bit to allow the endpoint to continue.
	IncompTx	0	<i>(High-bandwidth interrupt endpoints only)</i> This bit will be set if no response is received from the device to which the packet is being sent.
6	ClrDataTog	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to reset the endpoint data toggle to 0.

Table 218 • TX_CSRL_REG (Host) (continued)

Bit Number	Name	Reset Value	Function
5	RxStall	0	This bit is set when a STALL handshake is received. When this bit is set, any DMA request that is in progress is stopped, the FIFO is completely flushed and the TxPktRdy bit (bit 0 of this register) is cleared. The Cortex-M3 processor (or fabric master) should clear this bit.
4	SetupPkt	0	The Cortex-M3 processor (or fabric master) sets this bit at the same time the TxPktRdy bit (bit 0 of the register) is set, to send a SETUP token instead of an OUT token for the transaction. Setting this bit also clears the Data Toggle.
3	FlushFIFO	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to flush the latest packet from the endpoint TX FIFO. The FIFO pointer is reset, the TxPktRdy bit (bit0 of this register) is cleared and an interrupt is generated. May be set simultaneously with TxPktRdy to abort the packet that is currently being loaded into the FIFO. FlushFIFO should only be used when TxPktRdy is set. At other times, it may cause data to be corrupted. Also note that, if the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
2	Error	0	The controller sets this bit when 3 attempts have been made to send a packet and no handshake packet has been received. When the bit is set, an interrupt is generated, TxPktRdy (bit 0 of this register) is cleared and the FIFO is completely flushed. The Cortex-M3 processor (or fabric master) should clear this bit. Valid only when the endpoint is operating in Bulk or Interrupt mode.
1	FIFONotEmpty	0	The controller sets this bit when there is at least 1 packet in the transmit FIFO.
0	TxPktRdy	0	The Cortex-M3 processor (or fabric master) sets this bit after loading a data packet into the FIFO. It is cleared automatically when a data packet has been transmitted. An interrupt is also generated at this point (if enabled). TxPktRdy is also automatically cleared prior to loading a second packet into a double-buffered FIFO.

10.3.6.8 TX_CSRH_REG (in Peripheral mode) Bit Definitions

Table 219 • TX_CSRH_REG (Peripheral)

Bit Number	Name	Reset Value	Function
7	AutoSet	0	If the Cortex-M3 processor (or fabric master) sets this bit, TxPktRdy (bit 0 of TXCSRL_REG) will be set automatically when data of the maximum packet size (value in TxMaxP in TX_MAX_P_REG) is loaded into the transmit FIFO. If a packet of less than the maximum packet size is loaded, then TxPktRdy must be set manually. Should not be set for high-bandwidth ISO endpoints or high-bandwidth interrupt endpoints.
6	ISO	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the transmit endpoint for ISO transfers, and clears it to enable the transmit endpoint for bulk or interrupt transfers. This bit only has effect in Peripheral mode. In Host mode, it always returns zero.
5	Mode	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the endpoint direction as transmit and clears the bit to enable it as receive. This bit only has effect where the same endpoint FIFO is used for both transmit and receive transactions.

Table 219 • TX_CSRH_REG (Peripheral) (continued)

Bit Number	Name	Reset Value	Function
4	DMAReqEnab	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the DMA request for the transmit endpoint.
3	FrcDataTog	0	The Cortex-M3 processor (or fabric master) sets this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by interrupt transmit endpoints that are used to communicate rate feedback for ISO endpoints.
2	DMAReqMode	0	The Cortex-M3 processor (or fabric master) sets this bit to select DMA Request Mode 1 and clears it to select DMA Request Mode 0. This bit must not be cleared before or in the same cycle as the above DMAReqEnab bit (bit 4 of this register) is cleared.
[1:0]	Reserved	N/A	N/A

10.3.6.9 TX_CSRH_REG (in Host mode) Bit Definitions

Table 220 • TX_CSRH_REG (Host)

Bit Number	Name	Reset Value	Function
7	AutoSet	0	If the Cortex-M3 processor (or fabric master) sets this bit, TxPktRdy (bit0 of TXCSRL_REG) will be set automatically when data of the maximum packet size (value in TxMaxP of TX_MAX_P_REG) is loaded into the transmit FIFO. If a packet of less than the maximum packet size is loaded, then TxPktRdy must be set manually. Should not be set for high-bandwidth ISO endpoints or high-bandwidth interrupt endpoints.
6	Reserved	N/A	
5	Mode	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the endpoint direction as transmit and clears the bit to enable it as receive. This bit only has effect where the same endpoint FIFO is used for both transmit and receive transactions.
4	DMAReqEnab	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the DMA request for the transmit endpoint.
3	FrcDataTog	0	The Cortex-M3 processor (or fabric master) sets this bit to force the endpoint data toggle to switch and the data packet to be cleared from the FIFO, regardless of whether an ACK was received. This can be used by Interrupt transmit endpoints that are used to communicate rate feedback for ISO endpoints.
2	DMAReqMode	0	The Cortex-M3 processor (or fabric master) sets this bit to select DMA Request Mode 1 and clears it to select DMA Request Mode 0. This bit must not be cleared either before or in the same cycle as the above DMAReqEnab bit (bit4 of this register) is cleared.
1	Data Toggle Write Enable	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to enable the current state of the endpoint 0 data toggle to be written (see Data Toggle bit, below). This bit is automatically cleared once the new value is written.

Table 220 • TX_CSRH_REG (Host) (continued)

Bit Number	Name	Reset Value	Function
0	Data Toggle	0	When read, this bit indicates the current state of the endpoint 0 data toggle. If Data Toggle Write Enable (bit 1 of this register) is High, this bit may be written with the required setting of the data toggle. If Data Toggle Write Enable is Low, any value written to this bit is ignored.

10.3.6.10 RX_MAX_P_REG Bit Definitions

Table 221 • RX_MAX_P_REG

Bit Number	Name	Reset Value	Function
[15:11]	m-1	N/A	<p>If the core is configured for high-bandwidth ISO/interrupt endpoints or packet splitting on bulk endpoints, the register includes either 2 or 5 further bits that define a multiplier m, which is equal to one more than the value recorded.</p> <p>For bulk endpoints with the packet splitting option enabled, the multiplier m can be up to 32 and defines the number of USB packets of the specified payload which are to be amalgamated into a single data packet within the FIFO. If the packet splitting option is not enabled, bits[15:13] are not implemented and bits[12:11] (if included) are ignored.</p> <p>For ISO/interrupt endpoints operating in High speed mode and with the high-bandwidth option enabled, m may only be either 2 or 3 (corresponding to bit 11 set or bit 12 set) and it specifies the maximum number of such transactions that can take place in a single microframe. If either bit[11] or bit[12] is non-zero, the USB controller automatically combines the separate USB packets received in any microframe into a single packet within the receive FIFO. The maximum payload for each transaction is 1,024 bytes, so this allows up to 3,072 bytes to be received in each microframe. For ISO transfers in Full speed mode or if high-bandwidth is not enabled, bits[11] and [12] are ignored.</p> <p>The value written to bits[10:0] (multiplied by m in the case of high-bandwidth ISO transfers) must match the value given in the <i>wMaxPacketSize</i> field of the standard endpoint descriptor for the associated endpoint (refer to the <i>USB specification</i> revision 2.0, Chapter 9). A mismatch could cause unexpected results.</p> <p>The total amount of data represented by the value written to this register (specified payload $\times m$) must not exceed the FIFO size for the receive endpoint, and should not exceed half the FIFO size if double-buffering is required.</p>
[10:0]	RxMaxP	0	<p>Maximum payload/transaction. Maximum payload in bytes received in a single transaction. The value set can be up to 1,024 bytes but is subject to the constraints placed by the USB specification on packet sizes for bulk, interrupt, and ISO transfers in full speed and high speed operations.</p> <p>RxMaxP must be set to an even number of bytes for proper interrupt generation in DMA Mode 1.</p>

10.3.6.11 RX_CSRL_REG (in Peripheral mode) Bit Definitions

Table 222 • RX_CSRL_REG (Peripheral)

Bit Number	Name	Reset Value	Function
7	ClrDataTog	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to reset the endpoint data toggle to 0.
6	SentStall	0	This bit is set when a STALL handshake is transmitted. The Cortex-M3 processor (or fabric master) should clear this bit.
5	SendStall	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to issue a STALL handshake. The Cortex-M3 processor (or fabric master) clears this bit to terminate the stall condition. This bit has no effect where the endpoint is being used for ISO transfers.
4	FlushFIFO	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to flush the latest packet from the endpoint receive FIFO. The FIFO pointer is reset, the RxPktRdy bit (bit 0 of this register) is cleared. FlushFIFO should only be used when RxPktRdy is set. At other times, it may cause data to be corrupted. If the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
3	DataError	0	This bit is set when RxPktRdy (bit 0 of this register) is set if the data packet has a CRC or bit-stuff error. It is cleared when RxPktRdy is cleared. This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
2	OverRun	0	This bit is set if an OUT packet cannot be loaded into the receive FIFO. The Cortex-M3 processor (or fabric master) should clear this bit. This bit is only valid when the endpoint is operating in ISO mode. In Bulk mode, it always returns zero.
1	FIFOFull	0	This bit is set when no more packets can be loaded into the receive FIFO.
0	RxPktRdy	0	This bit is set when a data packet has been received. The Cortex-M3 processor (or fabric master) should clear this bit when the packet has been unloaded from the receive FIFO. An interrupt is generated when the bit is set.

10.3.6.12 RX_CSRL_REG (in Host mode) Bit Definitions

Table 223 • RX_CSRL_REG (Host)

Bit Number	Name	Reset Value	Function
7	ClrDataTog	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to reset the endpoint data toggle to 0.
6	RxStall	0	When a STALL handshake is received, this bit is set and an interrupt is generated. The Cortex-M3 processor (or fabric master) should clear this bit.
5	ReqPkt	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to request an IN transaction. It is cleared when RxPktRdy is set.

Table 223 • RX_CSRL_REG (Host) (continued)

Bit Number	Name	Reset Value	Function
4	FlushFIFO	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to flush the latest packet from the endpoint receive FIFO. The FIFO pointer is reset and the RxPktRdy bit (bit 0 of this register) is cleared. FlushFIFO should only be used when RxPktRdy is set. At other times, it may cause data to be corrupted. If the FIFO is double-buffered, FlushFIFO may need to be set twice to completely clear the FIFO.
3	DataError	0	When operating in ISO mode, this bit is set when RxPktRdy (bit 0 of this register) is set, if the data packet has a CRC or bit-stuff error, and cleared when RxPktRdy is cleared. In Bulk mode, this bit is set when the receive endpoint is halted, following the receipt of NAK responses for longer than the time set as the NAK limit by the RxInterval register. The Cortex-M3 processor (or fabric master) should clear this bit to allow the endpoint to continue. However, if double packet buffering is enabled, this alone will not allow the transfer to continue. In this case, the reqpkt bit should also be set in the same cycle this bit is cleared.
	NAK Timeout	0	
2	Error	0	The USB controller sets this bit when 3 attempts have been made to receive a packet and no data packet has been received. The Cortex-M3 processor (or fabric master) should clear this bit. This bit is only valid when the Rx endpoint is operating in Bulk or Interrupt mode. In ISO mode, it always returns zero.
1	FIFOFull	0	This bit is set when no more packets can be loaded into the receive FIFO.
0	RxPktRdy	0	This bit is set when a data packet has been received. The Cortex-M3 processor (or fabric master) should clear this bit when the packet has been unloaded from the receive FIFO. An interrupt is generated when the bit is set.

10.3.6.13 RX_CSRH_REG (in Peripheral mode) Bit Definitions

Table 224 • RX_CSRH_REG (Peripheral)

Bit Number	Name	Reset Value	Function
7	AutoClear	0	If the Cortex-M3 processor (or fabric master) sets this bit then the RxPktRdy bit (bit 0 in RXCSRL_REG) will be automatically cleared when a packet of RxMaxP (RX_MAX_P_REG) bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, RxPktRdy will have to be cleared manually. When using a DMA to unload the receive FIFO, data is read from the receive FIFO in 4-byte chunks, regardless of the RxMaxP. Therefore, the RxPktRdy bit will be cleared, as shown in Table 225 , page 326. Should not be set for high-bandwidth ISO endpoints.
6	ISO	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the receive endpoint for ISO transfers, and clears it to enable the receive endpoint for bulk/interrupt transfers.
5	DMAReqEnab	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the DMA request for the receive endpoint.

Table 224 • RX_CSRH_REG (Peripheral) (continued)

Bit Number	Name	Reset Value	Function
4	DisNyet	0	Bulk/interrupt transactions: the Cortex-M3 processor (or fabric master) sets this bit to disable the sending of NYET handshakes. When set, all successfully received packets are ACKed, including the point at which the FIFO becomes full. This bit only has effect in High speed mode, and should be set for all interrupt endpoints.
	PID Error	0	ISO transactions: the USB controller sets this bit to indicate a PID error in the received packet.
3	DMAReqMode	0	The Cortex-M3 processor (or fabric master) sets this bit to select DMA Request Mode 1 and clears it to select DMA Request Mode 0.
[2:1]	Reserved	N/A	
0	IncompRx	0	This bit is set in a high-bandwidth ISO/interrupt transfer if the packet in the receive FIFO is incomplete because parts of the data were not received. It is cleared when RxPktRdy (bit0 in RXCSRL_REG) is cleared. In anything other than ISO transfer, this bit will always return 0.

10.3.6.14 RxPktReady Bit Cleared

Table 225 • RxPktReady Bit Cleared

Remainder (RxMaxP/4)	Actual Bytes Read	Packet Sizes that will Clear RxPktRdy
0 (RxMaxP = 64 bytes)	RxMaxP	RxMaxP, RxMaxP – 1, RxMaxP – 2, RxMaxP – 3
3 (RxMaxP = 63 bytes)	RxMaxP + 1	RxMaxP, RxMaxP – 1, RxMaxP – 2
2 (RxMaxP = 62 bytes)	RxMaxP + 2	RxMaxP, RxMaxP – 1
1 (RxMaxP = 61 bytes)	RxMaxP + 3	RxMaxP

10.3.6.15 RX_CSRH_REG (in Host mode) Bit Definitions

Table 226 • RX_CSRH_REG (Host)

Bit Number	Name	Reset Value	Function
7	AutoClear	0	If the Cortex-M3 processor (or fabric master) sets this bit, the RxPktRdy bit (bit 0 in RXCSRL_REG) will be automatically cleared when a packet of RxMaxP (RX_MAX_P_REG) bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, RxPktRdy will have to be cleared manually. When using a DMA to unload the receive FIFO, data is read from the receive FIFO in 4-byte chunks regardless of the RxMaxP. Therefore, the RxPktRdy bit is cleared as shown in Table 225 on page 326 . Should not be set for high-bandwidth ISO endpoints.
6	AutoReq	0	If the Cortex-M3 processor (or fabric master) sets this bit, the ReqPkt bit (bit 5 in RX_CSRL_REG) will be set automatically when the RxPktRdy bit (bit 0 in RX_CSRL_REG) is cleared. This bit is automatically cleared when a short packet is received.
5	DMAReqEnab	0	The Cortex-M3 processor (or fabric master) sets this bit to enable the DMA request for the receive endpoint.

Table 226 • RX_CSRH_REG (Host) (continued)

Bit Number	Name	Reset Value	Function
4	PID Error	0	ISO transactions: The USB controller sets this bit to indicate a PID error in the received packet. Bulk/interrupt transactions: the setting of this bit is ignored.
3	DMAReqMode	0	The Cortex-M3 processor (or fabric master) sets this bit to select DMA Request mode 1 and clears it to select DMA Request mode 0.
2	Data Toggle Write Enable	0	The Cortex-M3 processor (or fabric master) writes a 1 to this bit to enable the current state of the endpoint0 data toggle to be written (refer to the Data Toggle bit, below). This bit is automatically cleared once the new value is written.
1	Data Toggle	0	When read, this bit indicates the current state of the endpoint0 data toggle. If Data Toggle Write Enable (bit[2] of this register) is high, this bit may be written with the required setting of the data toggle. If Data Toggle Write Enable is low, any value written to this bit is ignored.
0	IncompRx	0	This bit is set in a high-bandwidth ISO/interrupt transfer if the packet in the receive FIFO is incomplete because parts of the data were not received. It is cleared when RxPktRdy (bit 0 in RXCSRL_REG) is cleared. In anything other than ISO transfer, this bit always returns 0.

10.3.6.16 COUNT0_REG Bit Definitions

Table 227 • COUNT0_REG

Bit Number	Name	Reset Value	Function
[6:0]	Endpoint0 Rx Count	0	Indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change and is only valid while RxPktRdy (CSRL_REG.bit0) is set.

10.3.6.17 RX_COUNT_REG Bit Definitions

Table 228 • RX_COUNT_REG

Bit Number	Name	Reset Value	Function
[13:0]	Endpoint Rx Count	0	Holds the number of data bytes in the packet currently in line to be read from the receive FIFO. If the packet was transmitted as multiple bulk packets, the number given will be for the combined packet. The value returned changes as the FIFO is unloaded and is only valid while RxPktRdy (bit 0 in RX_CSRL_REG) is set.

10.3.6.18 TYPE0_REG (Host mode only)

Table 229 • TYPE0_REG

Bit Number	Name	Reset Value	Function
[7:6]	Speed	0	These bits should be written with the operating speed of the targeted device.
[5:0]	Reserved	N/A	

10.3.6.19 TX_TYPE_REG Bit Definitions

Table 230 • TX_TYPE_REG

Bit Number	Name	Reset Value	Function
[7:6]	Speed	0	Operating speed of the target device 00: Unused (If selected, the target is assumed to be using the same connection speed as the USB controller.) 01: High 10: Full 11: Low When the core is not configured with the multipoint option, these bits should not be accessed.
[5:4]	Protocol	0	The Cortex-M3 processor (or fabric master) should set this to select the required protocol for the transmit endpoint. 00: Control 01: ISO 10: Bulk 11: Interrupt
[3:0]	Target Endpoint Number	0	The Cortex-M3 processor (or fabric master) should set this value to the endpoint number contained in the transmit endpoint descriptor returned to the USB controller during device enumeration.

10.3.6.20 NAK_LIMIT0_REG (Hots mode only) Bit Definitions

Table 231 • NAK_LIMIT0_REG

Bit Number	Name	Reset Value	Function
[4:0]	Endpoint0 NAK Limit (m)	0	Sets the number of frames/microframes (high speed transfers) after which endpoint 0 should timeout on receiving a stream of NAK responses. Equivalent settings for other endpoints can be made through their TX_INTERVAL_REG and TX_INTERVAL_REG registers. The number of frames/microframes selected is $2(m - 1)$ (where m is the value set in the register, valid values are 2 – 16). If the host receives NAK responses from the target for more frames than the number represented by the limit set in this register, the endpoint will be halted. A value of 0 or 1 disables the NAK timeout function.

10.3.6.21 TX_INTERVAL_REG (Host mode only) Bit Definitions

Table 232 • TX_INTERVAL_REG

Bit Number	Name	Reset Value	Function
[7:0]	Tx Polling Interval/NAK Limit (m)	0	Defines the polling interval for the currently selected transmit endpoint for interrupt and ISO transfers. For bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses. There is one such register for each configured transmit endpoint (except endpoint 0). In each case the value that is set defines a number of frames/microframes (high speed transfers), as given in Table 233 , page 329.

10.3.6.22 Polling Intervals for Transfer Types

Table 233 • Polling Intervals for Transfer Types

Transfer Type	Speed	Valid Value (m)	Interpretation
Interrupt	Low speed or full speed	1 – 255	Polling interval is m frames.
	High speed	1 – 16	Polling interval is 2(m – 1) microframes.
Isochronous	Full speed or high speed	1 – 16	Polling interval is 2(m – 1) frames/microframes.
Bulk	Full speed or high speed	2 – 16	NAK Limit is 2(m – 1) frames/microframes. A value of 0 or 1 disables the NAK timeout function.

10.3.6.23 RX_TYPE_REG (Host mode only) Bit Definitions

Table 234 • RX_TYPE_REG

Bit Number	Name	Reset Value	Function
[7:6]	Speed	0	Operating speed of the target device: 00: Unused (If selected, the target is assumed to be using the same connection speed as the USB controller.) 01: High 10: Full 11: Low When the core is not configured with the multipoint option, these bits should not be accessed.
[5:4]	Protocol	0	The Cortex-M3 processor (or fabric master) should set this to select the required protocol for the receive endpoint: 00: Control 01: ISO 10: Bulk 11: Interrupt
[3:0]	Target Endpoint Number	0	The Cortex-M3 processor (or fabric master) should set this value to the endpoint number contained in the receive endpoint descriptor returned to the USB controller during device enumeration.

10.3.6.24 RX_INTERVAL_REG (Host mode only) Bit Definitions

Table 235 • RX_INTERVAL_REG

Bit Number	Name	Reset Value	Function
[7:0]	Rx Polling Interval/NAK Limit (m)	0	Defines the polling interval for the currently selected receive endpoint for Interrupt and ISO transfers. For bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving stream of NAK responses. There is one such register for each configured transmit endpoint (except endpoint 0). In each case the value that is set defines a number of frames/microframes (high speed transfers), as given in Table 233 on page 329 .

10.3.6.25 CONFIG_DATA_REG Bit Definitions

Table 236 • CONFIG_DATA_REG

Bit Number	Name	Reset Value	Function
7	MPRxE	N/A	When set to 1, automatic amalgamation of bulk packets is selected.
6	MPTxE	N/A	When set to 1, automatic splitting of bulk packets is selected.
5	BigEndian	N/A	Always 0. Indicates Little Endian ordering.
4	HBRxE	N/A	When set to 1, indicates high-bandwidth receive ISO endpoint support selected.
3	HBTxE	N/A	When set to 1, indicates high-bandwidth transmit ISO endpoint support selected.
2	DynFIFO Sizing	N/A	When set to 1, indicates dynamic FIFO sizing option selected.
1	SoftConE	N/A	Always 1. Indicates soft connect/disconnect.
0	UTMI DataWidth	N/A	Indicates selected UTMI+ data width. Always 0, indicating 8 bits.

10.3.6.26 FIFO_SIZE_REG Bit Definitions

Table 237 • FIFO_SIZE_REG

Bit Number	Name	Reset Value	Function
[7:4]	Rx FIFO Size	N/A	Returns the sizes of the FIFOs associated with the selected additional transmit/receive endpoints. The lower nibble, [3:0], encodes the size of the selected transmit endpoint FIFO; the upper nibble, [7:4], encodes the size of the selected receive endpoint FIFO. Values of 3 – 13 correspond to a FIFO size of 2n bytes (8 – 8,192 bytes). If an endpoint has not been configured, a value of 0 is displayed. When the transmit and receive endpoints share the same FIFO, the Rx FIFO size will be encoded as 0xF. The register only has this interpretation when the Index register is set to select one of endpoints 1 – 15 and dynamic sizing is not selected. It has a special interpretation when the INDEX_REG is set to select endpoint0; the result returned is not valid where dynamic FIFO sizing is used.
[3:0]	Tx FIFO Size	N/A	

10.3.7 FIFO Registers

These registers provide access to endpoint transmit and receive FIFOs. Writing to these addresses loads data into the transmit for the corresponding endpoint. Reading from these addresses unloads data from the receive FIFO for the corresponding endpoint.

Table 238 • FIFO Registers

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP0_FIFO_REG	0x0020	32	RW	0	Writing to this address loads data into the endpoint0 transmit FIFO. Reading from this address unloads data from the endpoint0 transmit FIFO.
EP1_FIFO_REG	0x0024	32	RW	0	Writing to this address loads data into the endpoint1 transmit FIFO. Reading from this address unloads data from the endpoint1 transmit FIFO.

Table 238 • FIFO Registers (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP2_FIFO_REG	0x0028	32	RW	0	Writing to this address loads data into the endpoint2 transmit FIFO. Reading from this address unloads data from the endpoint2 transmit FIFO.
EP3_FIFO_REG	0x002C	32	RW	0	Writing to this address loads data into the endpoint3 transmit FIFO. Reading from this address unloads data from the endpoint3 transmit FIFO.
EP4_FIFO_REG	0x0030	32	RW	0	Writing to this addresses loads data into the endpoint4 transmit FIFO. Reading from this address unloads data from the endpoint4 transmit FIFO.

10.3.7.1 EPx_FIFO_REG Bit Definitions

Table 239 • EPx_FIFO_REG (0x400430YZ)

Bit Number	Name	Reset Value	Function
[31:0]	Epx_TxRxDataAcc	0	Address of endpointx to write data into transmit FIFO or read from receive FIFO. The value of x can be 0, 1, 2, 3, or 4. The values of YZ would be differ with the register name: 1. EP0_FIFO_REG - 0x40043020 2. EP1_FIFO_REG - 0x40043024 3. EP2_FIFO_REG - 0x40043028 4. EP3_FIFO_REG - 0x4004302C 5. EP4_FIFO_REG - 0x40043030

10.3.8 Control and Status Registers (OTG, Dynamic FIFO and Version)

This section covers all registers in this category along with the address offset, functionality, and per bit details.

Table 240 • Additional Control and Status Registers (OTG, Dynamic FIFO, and Version)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
DEV_CTRL_REG (0x40043060)	0x0060	8	R	0x80	Selects whether the USB controller is operating in Peripheral mode or in Host mode, and for controlling and monitoring the USB VBus line. If the PHY is suspended, no PHY clock (XCLK) is received and the VBus is not sampled.
MISC_REG (0x40043061)	0x0061	8	R	0	Contains the early DMA enable bits for receive and transmit.
TX_FIFO_SIZE_REG (0x40043062)	0x0062	5	RW	0	Controls the size of the selected transmit endpoint FIFO.

Table 240 • Additional Control and Status Registers (OTG, Dynamic FIFO, and Version) (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
RX_FIFO_SIZE_REG (0x40043063)	0x0063	5	RW	0	Controls the size of the selected receive endpoint FIFO.
TX_FIFO_ADD_REG (0x40043064)	0x0064	14	RW	0	Controls the start address of the selected transmit endpoint FIFO.
RX_FIFO_ADD_REG (0x40043066)	0x0066	14	RW	0	Controls the start address of the selected receive endpoint FIFO.
VBUS_CSR_REG (write only) (0x40043068)	0x0068	4	W (VControl)	0	VControl is optionally a UTMI+ PHY vendor register. UTMI+ specification defines a 4-bit VControl register. The latency for the write (as measured between the positive edge of the CLK at the end of the AHB write cycle and the positive edge of XCLK when the UTMI+ PHY VControl register is loaded) will be between $H_c + 3X_c$ and $H_c + 4X_c$, where H_c is a cycle of CLK and X_c is a cycle of XCLK. The minimum period between successive writes to the VControl register must therefore be $H_c + 4X_c$ to ensure that the value is not corrupted while it is being synchronized to the XCLK domain.
VBUS_CSR_REG (read only) (0x40043068)	0x0068	8	R (VStatus)		VStatus is optionally a UTMI+ PHY vendor register. UTMI+ specification defines an 8-bit VStatus register. The VSTATUS input bus is sampled once every 6 XCLK cycles. The latency between the VSTATUS input bus from the PHY changing and the new value being read from the VStatus register (measured to the positive edge of CLK at the end of the AHB read cycle) will be between $2H_c + X_c$ and $3H_c + 6X_c$, where H_c is a cycle of CLK and X_c is a cycle of XCLK.
HW_VERSION_REG (0x4004306C)	0x006C	16	R		Returns information about the version of the USB controller. Specifically, the version of design (RTL) used to implement the USB controller. Useful for debug purposes.
Reserved	0x006E	N/A			

10.3.8.1 DEV_CTRL_REG Bit Definitions

Table 241 • DEV_CTRL_REG (0x40043060)

Bit Number	Name	Reset Value	Function
7	B-Device	1	Indicates whether the USB controller is operating as the A device or the B device. 0: A device 1: B device Only valid while a session is in progress. To determine the role when no session is in progress, set the session bit and read this bit. If the core is in Force_Host mode (a session has been started with TESTMODE_REG.bit[7] = 1), this bit indicates the state of the FAB_HOSTDISCON input signal.
6	FSDev	0	This read-only bit is set when a full speed or high speed device has been detected being connected to the port. High speed devices are distinguished from full speed by checking for high speed chirps when the device is reset. Only valid in Host mode.
5	LSDev	0	This read-only bit is set when a low speed device has been detected being connected to the port. Only valid in Host mode.
[4:3]	Vbus[1:0]	0	These read-only bits encode the current VBus level as given in Table 242 , page 334.
2	Host Mode	0	This read-only bit is set when the USB controller is acting as a host.
1	Host Req	0	When set, the USB controller initiates the host negotiation when Suspend mode is entered. It is cleared when host negotiation is completed. B device only).
0	Session	0	When operating as an A device, this bit is set or cleared by the Cortex-M3 processor (or fabric master) to start or end a session. When operating as a B device, this bit is set/cleared by the USB controller when a session starts/ends. It is also set by the Cortex-M3 processor (or fabric master) to initiate the session request protocol. When the USB controller is in Suspend mode, the bit may be cleared by the Cortex-M3 processor (or fabric master) to perform a software disconnect. Clearing this bit when the core is not suspended will result in undefined behavior.

10.3.8.2 Encoding of VBus Level

Table 242 • Encoding of VBus Level

D4	D3	Meaning
0	0	Below session end
0	1	Above session end, below Avalid
1	0	Above Avalid, below VBus valid
1	1	Above VBus valid

10.3.8.3 MISC_REG Bit Definitions

Table 243 • MISC_REG (0x40043061)

Bit Number	Name	Reset Value	Function
1	TX_EDMA	0	0: DMA_REQ signal for all IN endpoints is deasserted when MAXP (TX_MAX_P_REG) bytes have been written to an endpoint. This is late mode. 1: DMA_REQ signal for all IN endpoints is deasserted when MAXP-8 (TX_MAX_P_REG-8) bytes have been written to an endpoint. This is early mode.
0	RX_EDMA	0	0: DMA_REQ signal for all OUT endpoints is deasserted when MAXP (TX_MAX_P_REG) bytes have been read to an endpoint. This is late mode. 1: DMA_REQ signal for all OUT endpoints is deasserted when MAXP-8 (TX_MAX_P_REG-8) bytes have been read to an endpoint. This is early mode.

10.3.8.4 TX_FIFO_SIZE_REG Bit Definitions

Table 244 • TX_FIFO_SIZE_REG (0x40043062)

Bit Number	Name	Reset Value	Function
4	DPB	0	Defines whether double-packet buffering is supported. When '1', double-packet buffering is supported. When '0', only single-packet buffering is supported.
[3:0]	SZ[3:0]	0	Maximum packet size to be allowed for (before any splitting within the FIFO of bulk/high-bandwidth packets prior to transmission). 00 00: 8 Bytes 0 001: 16 Bytes 0010: 32 Bytes 0011: 64 Bytes 0100: 128 Bytes 0101: 256 Bytes 0110: 512 Bytes 0111: 1,024 Bytes 1000: 2,048 Bytes 1001: 4,096 Bytes If DPB = 0, the FIFO will also be this size; if DPB = 1, the FIFO will be twice this size.

10.3.8.5 RX_FIFO_SIZE_REG Bit Definitions

Table 245 • RX_FIFO_SIZE_REG (0x40043063)

Bit Number	Name	Reset Value	Function
4	DPB	0	Defines whether double-packet buffering is supported. When '1', double-packet buffering is supported. When '0', only single-packet buffering is supported.
[3:0]	SZ[3:0]	0	Maximum packet size to be allowed for (before any splitting within the FIFO of bulk/high-bandwidth packets prior to transmission). 00 00: 8 Bytes 0 001: 16 Bytes 0010: 32 Bytes 0011: 64 Bytes 0100: 128 Bytes 0101: 256 Bytes 0110: 512 Bytes 0111: 1,024 Bytes 1000: 2,048 Bytes 1001: 4,096 Bytes If DPB = 0, the FIFO will also be this size; if DPB = 1, the FIFO will be twice this size.

10.3.8.6 TX_FIFO_ADD_REG Bit Definitions

Table 246 • TX_FIFO_ADD_REG (0x40043064)

Bit Number	Name	Reset Value	Function
13	Reserved	N/A	
[12:0]	AD[12:0]	0	Start address of the transmit endpoint FIFO in units of 8 bytes as given in Table 247 , page 336.

10.3.8.7 Start Address of Transmit Endpoint

Table 247 • Start Address of Transmit Endpoint

AD[12:0]					Start Address
0	0	...	0	0	0x0000
0	0	...	0	1	0x0008
0	0	...	1	0	0x0010
	
1	1	...	1	1	0xFFFF8

10.3.8.8 RX_FIFO_ADD_REG Bit Definitions

Table 248 • RX_FIFO_ADD_REG (0x40043066)

Bit Number	Name	Reset Value	Function
13	Reserved	N/A	
[12:0]	AD[12:0]	0	Start address of the receive endpoint FIFO in units of 8 bytes as given in Table 247 , page 336.

10.3.8.9 VBUS_CSR_REG (write only control) Bit Definitions

Table 249 • VBUS_CSR_REG (write only) (0x40043068)

Bit Number	Name	Reset Value	Function
[3:0]	Vcontrol	0	Vendor-specific control data

10.3.8.10 VBUS_CSR_REG (read only status) Bit Definitions

Table 250 • VBUS_CSR_REG (read only) (0x40043068)

Bit Number	Name	Reset Value	Function
[7:0]	Vstatus	0	Vendor-specific status data

10.3.8.11 HW_VERSION_REG Bit Definitions

Table 251 • HW_VERSION_REG (0x4004306C)

Bit Number	Name	Reset Value	Function
15	Reserved	N/A	
[14:10]	xx	0	Major version number (range 0–31)
[9:0]	yyy	0	Minor version number (range 0–999)

10.3.9 ULPI and Configuration Registers

These registers correspond to the ULPI interface and link specific. This section covers all registers in this category along with the address offset, functionality, and per bit details.

Table 252 • ULPI and Configuration Registers

Register Name	Address	Width	R/W Type	Reset Value	Description
ULPI_VBUS_CTRL_REG (0x40043070)	0x40043070	8	R	0	ULPI PHYs can use an external charge pump to generate VBus rather than an internal charge pump. This register allows selection of the external charge pump. It also allows this selection to be displayed through an external VBus indicator. This register is read back from the PHY clock domain. These bits do not therefore return updated values while the PHY is suspended.
ULPI_CARKit_CTRL_REG (0x40043071)	0x40043071	8	R	0	Provides the basic control needed by ULPI-compatible PHYs when interfacing to in-car CarKit systems.
ULPI_IRQ_MASK_REG (0x40043072)	0x40043072	8	R	0	Enables the assertion of MC_NINT in response to the possible interrupt sources.
ULPI_IRQ_SRC_REG (0x40043073)	0x40043073	8	R	0	This register shows the unmasked value of the possible interrupt sources.
ULPI_DATA_REG (0x40043074)	0x40043074	8	R	0	Contains the data associated with register reads/writes conducted through the ULPI interface.
ULPI_ADDR_REG (0x40043075)	0x40043075	8	R	0	Contains the address of the register being read/written through the ULPI interface.
ULPI_REG_CTRL (0x40043076)	0x40043076	8	R	0	Contains control and status bits relating to the register being read/written through the ULPI interface.
ULPI_RAW_DATA_REG (0x40043077) (Asynchronous) ULPI_RAW_DATA_REG (0x40043077) (Synchronous)	0x40043077	8	R	0	This register is used in Asynchronous modes to sample the ULPI bus and in Synchronous mode to store the last received command.
EP_INFO_REG (0x40043078)	0x40043078	8	R	0	Allows read-back of the number of transmit and receive endpoints included in the design.
RAM_INFO_REG (0x40043079)	0x40043079	8	R	0	Provides information about the number of DMA channels and the width of the RAM.
LINK_INFO_REG (0x4004307A)	0x4004307A	8	RW	0x5C	Allows configuration of link-specific delays
VP_LEN_REG (0x4004307B)	0x4004307B	8	RW	0x3C	This register allows setting duration of the Vbus pulsing charge
HS_EOF1_REG (0x4004307C)	0x4004307C	8	RW	0x80	Sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for high speed transactions.

Table 252 • ULPI and Configuration Registers (continued)

Register Name	Address	Width	R/W Type	Reset Value	Description
FS_EOF1_REG (0x4004307D)	0x4004307D	8	RW	0x77	Sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for full speed transactions.
LS_EOF1_REG (0x4004307E)	0x4004307E	8	RW	0x72	Sets the minimum time gap that is to be allowed between the start of the last transaction and the EOF for low speed transactions.
SOFT_RESET_REG (0x4004307F)	0x4004307F	8	RW	0	Asserts the output reset signals, NRSTO and NRSTOX, low. This register is self clearing and is reset by the input NRST.

10.3.9.1 ULPI_VBUS_CTRL_REG Bit Definitions

Table 253 • ULPI_VBUS_CTRL_REG (0x40043070)

Bit Number	Name	Reset Value	Function
[7:2]	Reserved	N/A	
1	UseExtVbusInd	0	When 1, selects the use of an external VBus indicator (overcurrent indicator) in the PHY's VbusState determination.
0	UseExtVbus	0	When 1, selects the use of an external charge pump.

10.3.9.2 ULPI_CARKIT_CTRL_REG Bit Definitions

Table 254 • ULPI_CARKIT_CTRL_REG (0x40043071)

Bit Number	Name	Reset Value	Function
[7:6]	Reserved	N/A	Reserved/ N/A
5	CarKitActiveEnd	0	Set by link when CarKitActive (bit 1 of this register) is cleared. This bit must be cleared by software. Signifies that the USB controller's (synchronous) USB mode has been entered.
4	RxCmdEvent	0	Set by link when a RxCmd has been latched. This bit must be cleared by software.
3	CancelCarKit	0	Set by software to abort CarKit mode and wake up the PHY. This bit auto-clears when the PHY enters Synchronous mode.
2	AltIntEvent	0	Set by link when an alt_int (ULPI_RAW_DATA_REG.bit7) event occurs. This bit must be cleared by software.
1	CarKitActive	0	Set by link when the (asynchronous) CarKit mode is entered after DIR goes high. It is cleared on the falling edge of DIR.
0	DisableUTMI	0	Set and cleared by software to decouple the reconstituted UTMI signals from the USB controller prior to entering CarKit mode. Can not be cleared while ULPICarKitControl CarKitActive bit (bit 1 of this register) is set.

10.3.9.3 ULPI_IRQ_MASK_REG Bit Definitions

Table 255 • ULPI_IRQ_MASK_REG (0x40043072)

Bit Number	Name	Reset Value	Function
[7:4]	Reserved	N/A	
3	RxCmdIntEn	0	Assert MC_NINT if RxCmdInt (ULPI_IRQ_SRC_REG.bit3) is set. To clear MC_NINT, the software must clear RxCmdEvent (ULPI_CARKIT_CTRL_REG.bit4).
2	ActiveEndIntEn	0	Assert MC_NINT if ActiveEndInt (ULPI_IRQ_SRC_REG.bit2) is set. To clear MC_NINT, the software must clear CarKitActiveEnd (ULPI_CARKIT_CTRL_REG.bit5).
1	AltIntEn	0	Assert MC_NINT if AltInt (ULPI_IRQ_SRC_REG.bit1) is set. To clear MC_NINT, the software must clear AltIntEvent (ULPI_CARKIT_CTRL_REG.bit2).
0	RegIntEn	0	Assert MC_NINT if RegInt (ULPI_IRQ_SRC_REG.bit0) is set. To clear MC_NINT, the software must clear ULPIRegCmpl (ULPI_REG_CTRL_REG.bit1).

10.3.9.4 ULPI_IRQ_SRC_REG Bit Definitions

Table 256 • ULPI_IRQ_SRC_REG (0x40043073)

Bit Number	Name	Reset Value	Function
[7:4]	Reserved	N/A	
3	RxCmdInt	0	Asserted if RxCmdEvent (ULPI_CARKIT_CTRL_REG.bit4) is set. To clear the interrupt, the software must clear RxCmdEvent.
2	ActiveEndInt	0	Asserted if CarKitActiveEnd (ULPI_CARKIT_CTRL_REG.bit5) is set. To clear the interrupt, the software must clear CarKitActiveEnd.
1	AltInt	0	Asserted if AltIntEvent (ULPI_CARKIT_CTRL_REG.bit2) is set. To clear the interrupt, the software must clear AltIntEvent.
0	RegInt	0	Asserted if ULPIRegCmpl (ULPI_REG_CTRL_REG.bit1) is set. To clear the interrupt, the software must clear ULPIRegCmpl.

10.3.9.5 ULPI_DATA_REG Bit Definitions

Table 257 • ULPI_DATA_REG (0x40043074)

Bit Number	Name	Reset Value	Function
[7:0]	ULPIRegData	0	Register data for PHY register access

10.3.9.6 ULPI_ADDR_REG Bit Definitions

Table 258 • ULPI_ADDR_REG (0x40043075)

Bit Number	Name	Reset Value	Function
[7:0]	ULPIRegAddr	0	Address for PHY register access

10.3.9.7 ULPI_REG_CTRL Bit Definitions

Table 259 • ULPI_REG_CTRL (0x40043076)

Bit Number	Name	Reset Value	Function
[7:3]	Reserved	N/A	
2	ULPIRdnWr	0	Set by software for register read access. Cleared by software for register write access.
1	ULPIRegCmplt	0	Set by link when register access is complete. This bit must be cleared by software.
0	ULPIRegReq	0	Set by software to initiate register access. This is cleared when ULPIRegCmplt (bit 1 of this register) is set.

10.3.9.8 ULPI_RAW_DATA_REG (Asynchronous Mode) Bit Definitions

When one of the PHY's Asynchronous modes is selected, this register is used to indicate the present value of the ULPI bus, latched by any transition on int (on data(3)).

Table 260 • ULPI_RAW_DATA_REG (0x40043077) (Asynchronous)

Bit Number	Name	Reset Value	Function
[7:4]	Reserved	N/A	
3	data(3)	0	Active high interrupt indication (int)
2	data(2)	0	Single-ended zero (se0)
1	data(1)	0	Differential data (dat)
0	data(0)	0	Active high transmit enable (tx_enable)

10.3.9.9 ULPI_RAW_DATA_REG (Synchronous mode) Bit Definitions

When the PHY's Synchronous mode is used, this register is used to store the last RxCmd.

Because RxCmds are received in the PHY clock domain and this register is read in the CPU clock domain, RxCmds may be missed if they are concurrent and the CPU clock is slower than the PHY clock. This should not be an issue for CarKit negotiation because this process lasts for over 1 ms, but this feature should not be relied upon to monitor transactions carried out at USB bus speed.

Table 261 • ULPI_RAW_DATA_REG (0x40043077) (Synchronous)

Bit Number	Name	Reset Value	Function
7	alt_int	0	Asserted when a non-USB interrupt occurs. In particular, it must be set if an unmasked event occurs on any bit of the PHY's CarKit Interrupt Latch register.
6	ID	0	Set to the value of the IDDIG (valid 50 ms after IDPULLUP is asserted).
[5:4]	RxEvent[1:0]	0	Encoded UTMI event signals are given in Table 262 , page 342.
[3:2]	VbusState[1:0]	0	Encoded Vbus voltage state. 00: Vbus < VB_Sess_END 01: VB_Sess_END <= Vbus < VB_Sess_VLD . . 10: VB_Sess_VLD <= Vbus < VB_Vbus_VLD 11: VB_Sess_VLD <= Vbus

Table 261 • ULPI_RAW_DATA_REG (0x40043077) (Synchronous) (continued)

Bit Number	Name	Reset Value	Function
[1:0]	LineState[1:0]	0	UTMI+ LineState signals

10.3.9.10 Encoded UTMI Event Signals

Table 262 • Encoded UTMI Event Signals

Value	RxActive	RxError	Host Disconnect
00	0	0	0
01	1	0	0
10	1	1	0
11	X	X	1

10.3.9.11 EP_INFO_REG Bit Definitions

Table 263 • EP_INFO_REG (0x40043078)

Bit Number	Name	Reset Value	Function
[7:4]	RxEndPoints	0	The number of receive endpoints implemented in the design.
[3:0]	TxEndPoints	0	The number of transmit endpoints implemented in the design.

10.3.9.12 RAM_INFO_REG Bit Definitions

Table 264 • RAM_INFO_REG (0x40043079)

Bit Number	Name	Reset Value	Function
[7:4]	DMACHans	0	The number of DMA channels implemented in the design
[3:0]	RamBits	0	The width of the RAM address bus

10.3.9.13 LINK_INFO_REG Bit Definitions

Table 265 • LINK_INFO_REG (0x4004307A)

Bit Number	Name	Reset Value	Function
[7:4]	WTCON	0x5	Sets the wait to be applied to allow for the connect/disconnect filter in units of 533.3 ns. The default setting corresponds to 2.667 μ s.
[3:0]	WTID	0xC	Sets the delay to be applied from IDPULLUP being asserted to IDDIG being considered valid in units of 4.369 ms. The default setting corresponds to 52.43 ms.

10.3.9.14 VP_LEN_REG Bit Definitions

Table 266 • VP_LEN_REG (0x4004307B)

Bit Number	Name	Reset Value	Function
[7:0]	VPLEN	0x3C	Sets the duration of the VBus pulsing charge in units of 546.1 μ s. The default setting corresponds to 32.77 ms.

10.3.9.15 HS_EOF1_REG Bit Definitions

Table 267 • HS_EOF1_REG (0x4004307C)

Bit Number	Name	Reset Value	Function
[7:0]	HS_EOF1	0x80	For high speed transactions: Sets the time before EOF to stop beginning new transactions, in units of 133.3 ns. The default setting corresponds to 17.07 μ s.

10.3.9.16 FS_EOF1_REG Bit Definitions

Table 268 • FS_EOF1_REG (0x4004307D)

Bit Number	Name	Reset Value	Function
[7:0]	FS_EOF1	0x77	For full speed transactions: Sets the time before EOF to stop beginning new transactions, in units of 533.3 ns. The default setting corresponds to 63.46 μ s.

10.3.9.17 LS_EOF1_REG Bit Definitions

Table 269 • LS_EOF1_REG (0x4004307E)

Bit Number	Name	Reset Value	Function
[7:0]	LS_EOF1	0x72	For low speed transactions: Sets the time before EOF to stop beginning new transactions, in units of 1.067 μ s. The default setting corresponds to 121.6 μ s.

10.3.9.18 SOFT_RESET_REG Bit Definitions

Table 270 • SOFT_RESET_REG (0x4004307F)

Bit Number	Name	Reset Value	Function
[7:2]	Reserved	N/A	
1	NRSTX	0	The default value of this bit is 0. When a 1 is written to this bit, the output NRSTXO will be asserted (low) within a minimum delay of 7 cycles of the CLK input. The output NRSTXO is asynchronously asserted and synchronously deasserted with respect to XCLK. This register is self clearing and is reset by the input NRST.
0	NRST	0	The default value of this bit is 0. When a 1 is written to this bit, the output NRSTO is asserted (Low) within a minimum delay of 7 cycles of the CLK input. The output NRSTO is asynchronously asserted and synchronously deasserted with respect to CLK. This register is self clearing and will be reset by the input NRST.

10.3.10 Non-Indexed End Point Control/Status Registers

The registers available at 10h–1Fh are accessible independently of the setting of the Index register. 100h–10Fh for EP0 registers; 110h–11Fh for EP1 registers; 120h–12Fh for EP2; and so on until EP4. For each set, a separate table for registers is included. Since all registers are similar except for address offset, a common bit definition table is included for each register.

10.3.10.1 Endpoint0 Control and Status Registers

Table 271 • Endpoint0 Control and Status Registers

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP0_TX_MAX_P_REG	0x0100	16	RW	0	Maximum packet size for host transmit endpoint0
EP0_TX_CSR_REG	0x0102	16	R	0	Provides control and status bits for transmit endpoint0.
EP0_RX_MAX_P_REG	0x0104	16	RW	0	Defines the maximum amount of data that can be transferred through receive endpoint0 in a single operation.
EP0_RX_CSR_REG	0x0106	16	R	0	Provides control and status bits for transfers through the receive endpoint0.
EP0_RX_COUNT_REG	0x0108	16	R	0	Holds the number of data bytes in the packet currently in line to be read from the endpoint0 receive FIFO. If the packet was transmitted as multiple bulk packets, the number given will be for the combined packet.
EP0_TX_TYPE_REG	0x010A	8	W	0	Reads the number of bytes from peripheral endpoint0 transmit FIFO
EP0_TX_INTERVAL_REG	0x010B	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host transmit endpoint0.
EP0_RX_TYPE_REG	0x010C	8	RW	0	Sets the transaction protocol, speed, and peripheral endpoint number for the host receive endpoint0.
EP0_RX_INTERVAL_REG	0x010D	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host receive endpoint0.
EP0_FIFO_SIZE_REG	0x010E	8	R		Returns the configured size of the endpoint0 receive FIFO and transmit FIFOs.

10.3.10.2 Endpoint1 Control and Status Registers

Table 272 • Endpoint1 Control and Status Registers

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP1_TX_MAX_P_REG	0x0110	16	R	0	Maximum packet size for host transmit endpoint1.
EP1_TX_CSR_REG	0x0112	16	R	0	Provides control and status bits for transmit endpoint1.
EP1_RX_MAX_P_REG	0x0114	16	RW	0	Defines the maximum amount of data that can be transferred through receive endpoint1 in a single operation.
EP1_RX_CSR_REG	0x0116	16	R	0	Provides control and status bits for transfers through the receive endpoint1.
EP1_RX_COUNT_REG	0x0118	16	R	0	Holds the number of data bytes in the packet currently in line to be read from the endpoint1 receive FIFO. If the packet is transmitted as multiple bulk packets, the number given will be for the combined packet.
EP1_TX_TYPE_REG	0x011A	8	W	0	Reads the number of bytes to be read from peripheral endpoint1 transmit FIFO
EP1_TX_INTERVAL_REG	0x011B	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host transmit endpoint1.
EP1_RX_TYPE_REG	0x011C	8	RW	0	Sets the transaction protocol, speed, and peripheral endpoint number for the host receive endpoint1.
EP1_RX_INTERVAL_REG	0x011D	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host receive endpoint1.
EP1_FIFO_SIZE_REG	0x011E	8	R		Returns the configured size of the endpoint1 receive FIFO and transmit FIFOs.

10.3.10.3 Endpoint2 Control and Status Registers

Table 273 • Endpoint2 Control and Status Registers

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP2_TX_MAX_P_REG	0x0120	16	RW	0	Maximum packet size for host transmit endpoint2.
EP2_TX_CSR_REG	0x0122	16	R	0	Provides control and status bits for transmit endpoint2.
EP2_RX_MAX_P_REG	0x0124	16	RW	0	Defines the maximum amount of data that can be transferred through receive endpoint2 in a single operation.

Table 273 • Endpoint2 Control and Status Registers (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP2_RX_CSR_REG	0x0126	16	R	0	Provides control and status bits for transfers through the receive endpoint2.
EP2_RX_COUNT_REG	0x0128	16	R	0	Holds the number of data bytes in the packet currently in line to be read from the endpoint2 receive FIFO. If the packet is transmitted as multiple bulk packets, the number given will be for the combined packet.
EP2_TX_TYPE_REG	0x012A	8	W	0	Reads the number of bytes from peripheral endpoint2 transmit FIFO.
EP2_TX_INTERVAL_REG	0x012B	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host transmit endpoint2.
EP2_RX_TYPE_REG	0x012C	8	RW	0	Sets the transaction protocol, speed, and peripheral endpoint number for the host receive endpoint2.
EP2_RX_INTERVAL_REG	0x012D	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host receive endpoint2.
EP2_FIFO_SIZE_REG	0x012E	8	R		Returns the configured size of the endpoint2 receive FIFO and transmit FIFOs.

10.3.10.4 Endpoint3 Control and Status Registers

Table 274 • Endpoint3 Control and Status Registers

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP3_TX_MAX_P_REG	0x0130	16	RW	0	Maximum packet size for host transmit endpoint3.
EP3_TX_CSR_REG	0x0132	16	R	0	Provides control and status bits for transmit endpoint3.
EP3_RX_MAX_P_REG	0x0134	16	RW	0	Defines the maximum amount of data that can be transferred through receive endpoint3 in a single operation.
EP3_RX_CSR_REG	0x0136	16	R	0	Provides control and status bits for transfers through the receive endpoint3.
EP3_RX_COUNT_REG	0x0138	16	R	0	Holds the number of data bytes in the packet currently in line to be read from the endpoint3 receive FIFO. If the packet is transmitted as multiple bulk packets, the number given will be for the combined packet.
EP3_TX_TYPE_REG	0x013A	8	W	0	Reads the number of bytes from peripheral endpoint3 transmit FIFO.

Table 274 • Endpoint3 Control and Status Registers (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP3_TX_INTERVAL_REG	0x013B	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host transmit endpoint3.
EP3_RX_TYPE_REG	0x013C	8	RW	0	Sets the transaction protocol, speed, and peripheral endpoint number for the host receive endpoint3.
EP3_RX_INTERVAL_REG	0x013D	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host receive endpoint3.
EP3_FIFO_SIZE_REG	0x013E	8	R		Returns the configured size of the endpoint3 receive FIFO and transmit FIFOs.

10.3.10.5 Endpoint4 Control and Status Registers

Table 275 • Endpoint4 Control and Status Registers

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP4_TX_MAX_P_REG	0x0140	16	RW	0	Maximum packet size for host transmit endpoint4.
EP4_TX_CSR_REG	0x0142	16	R	0	Provides control and status bits for transmit endpoint4.
EP4_RX_MAX_P_REG	0x0144	16	RW	0	Defines the maximum amount of data that can be transferred through receive endpoint4 in a single operation.
EP4_RX_CSR_REG	0x0146	16	R	0	Provides control and status bits for transfers through the receive endpoint4.
EP4_RX_COUNT_REG	0x0148	16	R	0	Holds the number of data bytes in the packet currently in line to be read from the endpoint4 receive FIFO. If the packet is transmitted as multiple bulk packets, the number given will be for the combined packet.
EP4_TX_TYPE_REG	0x014A	8	W	0	Reads the number of bytes from peripheral endpoint4 transmit FIFO.
EP4_TX_INTERVAL_REG	0x014B	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host transmit endpoint4.
EP4_RX_TYPE_REG	0x014C	8	RW	0	Sets the transaction protocol, speed, and peripheral endpoint number for the host receive endpoint4.
EP4_RX_INTERVAL_REG	0x014D	8	RW	0	Sets the polling interval for interrupt/ISOC transactions or the NAK response timeout on bulk transactions for host receive endpoint4.
EP4_FIFO_SIZE_REG	0x014E	8	R		Returns the configured size of the endpoint4 receive FIFO and transmit FIFOs.

10.3.10.6 EPx_TX_MAX_P_REG Bit Definitions

Table 276 • EPx_TX_MAX_P_REG

Bit Number	Name	Reset Value	Function
[15:11]	Reserved	N/A	
[10:0]	EPx_TxMaxP	0	Maximum payload/transaction Maximum payload in bytes transmitted in a single transaction. The value set can be up to 1,024 bytes but is subject to the constraints placed by the USB specification on packet sizes for bulk, interrupt, and ISO transfers in full speed and high speed operations. This must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

10.3.10.7 EPx_RX_COUNT_REG Bit Definitions

Table 277 • EPx_RX_COUNT_REG

Bit Number	Name	Reset Value	Function
[13:0]	EPx_Rx Count	0	Holds the number of data bytes in the packet currently in line to be read from the receive FIFO. If the packet is transmitted as multiple bulk packets, the number given will be for the combined packet. The value returned changes as the FIFO is unloaded and is only valid while RxPktRdy (bit0 in RX_CSRL_REG) is set.

10.3.10.8 EPx_RX_MAX_P_REG Bit Definitions

Table 278 • EPx_RX_MAX_P_REG

Bit Number	Name	Reset Value	Function
[15:11]	Reserved	N/A	
[10:0]	EPx_RxMaxP	0	Maximum Payload/transaction Maximum payload in bytes received in a single transaction. The value set can be up to 1,024 bytes but is subject to the constraints placed by the USB specification on packet sizes for bulk, interrupt, and ISO transfers in full speed and high speed operations. This must be set to an even number of bytes for proper interrupt generation in DMA Mode 1.

10.3.10.9 EPx_TX_TYPE_REG Bit Definitions

Table 279 • EPx_TX_TYPE_REG

Bit Number	Name	Reset Value	Function
[7:6]	EPx_Speed	0	Operating speed of the target device: 00: Unused If selected, the target is assumed to be using the same connection speed as the USB controller. 01: High 10: Full 11: Low When the core is not configured with the multipoint option, these bits should not be accessed.
[5:4]	EPx_Protocol	0	The Cortex-M3 processor (or fabric master) should set this to select the required protocol for the transmit endpoint: 00: Control 01: ISO 10: Bulk 11: Interrupt
[3:0]	EPx_Target Endpoint Number	0	The Cortex-M3 processor (or fabric master) should set this value to the endpoint number contained in the transmit endpoint descriptor returned to the USB controller during device enumeration.

10.3.10.10 EPx_TX_INTERVAL_REG Bit Definitions

Table 280 • EPx_TX_INTERVAL_REG

Bit Number	Name	Reset Value	Function
[7:0]	EPx_Tx Polling Interval / NAK Limit (m)	0	Defines the polling interval for endpointx transmit for interrupt and ISO transfers. For bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving a stream of NAK responses. The value that is set defines number of frames/microframes (high speed transfers), as given in Table 233 , page 329.

10.3.10.11 EPx_RX_TYPE_REG Bit Definitions

Table 281 • EPx_RX_TYPE_REG

Bit Number	Name	Reset Value	Function
[7:6]	EPx_Speed	0	Operating speed of the target device: 00: Unused (If selected, the target is assumed to be using the same connection speed as the USB controller.) 01: High 10: Full 11: Low When the core is not configured with the multipoint option, these bits should not be accessed.

Table 281 • EPx_RX_TYPE_REG (continued)

Bit Number	Name	Reset Value	Function
[5:4]	EPx_Protocol	0	The Cortex-M3 processor (or fabric master) should set this to select the required protocol for the receive endpoint: 00: Control 01: ISO 10: Bulk 11: Interrupt
[3:0]	EPx_Target Endpoint Number	0	The Cortex-M3 processor (or fabric master) should set this value to the endpoint number contained in the receive endpoint descriptor returned to the USB controller during device enumeration.

10.3.10.12EPx_RX_INTERVAL_REG Bit Definitions

Table 282 • EPx_RX_INTERVAL_REG

Bit Number	Name	Reset Value	Function
[7:0]	EPx_Rx Polling Interval / NAK Limit (m)	0	Defines the polling interval receive endpointx for interrupt and ISO transfers. For bulk endpoints, this register sets the number of frames/microframes after which the endpoint should timeout on receiving stream of NAK responses. The value that is set defines a number of frames/microframes (high speed transfers), as given in Table 233 , page 329.

10.3.10.13EPx_FIFO_SIZE_REG Bit Definitions

Table 283 • EPx_FIFO_SIZE_REG

Bit Number	Name	Reset Value	Function
[7:4]	EPx_Rx FIFO Size	N/A	Returns the sizes of the FIFOs associated with endpointx. The lower nibble, [3:0], encodes the size of the transmit FIFO; the upper nibble, [7:4], encodes the size of the receive endpoint FIFO. Values of 3 – 13 correspond to a FIFO size of 2 ⁿ bytes (8 – 8192 bytes). If an endpoint has not been configured, a value of 0 will be displayed. When the transmit and receive endpoints share the same FIFO, the Rx FIFO size will be encoded as 0xF. The register only has this interpretation when the Index register is set to select one of endpoints 1 – 15 and dynamic sizing is not selected. It has a special interpretation when the INDEX_REG is set to select endpoint 0; the result returned is not valid where dynamic FIFO sizing is used.
[3:0]	EPx_Tx FIFO Size	N/A	

10.3.11 Extended Registers

These registers correspond to extended register set. This section covers all registers in this category along with the address offset, functionality, and per bit details.

Table 284 • Extended Registers Description

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EPx_RQ_PKT_COUNT_REG (0x40043XYZ)					This read/write register is used in Host mode to specify the number of packets that are to be transferred in a block transfer of one or more bulk packets of length MaxP (EPx_RX_MAX_P_REG) to receive endpointx. The core uses the value recorded in this register to determine the number of requests to issue where the AutoReq option (RX_CSRH_REG.bit 6 has been set.
EP0_RQ_PKT_COUNT_REG	0x0300	16	RW	0	
EP1_RQ_PKT_COUNT_REG	0x0304	16	RW	0	
EP2_RQ_PKT_COUNT_REG	0x0308	16	RW	0	
EP3_RQ_PKT_COUNT_REG	0x030C	16	RW	0	
EP4_RQ_PKT_COUNT_REG	0x0310	16	RW	0	
RX_DPKT_BUF_DIS_REG (0x40043340)	0x0340	16	RW	0	Indicates which of the receive endpoints (EP0, EP1, EP2, EP3, EP4) have disabled the double packet buffer functionality.
TX_DPKT_BUF_DIS_REG (0x40043342)	0x0342	16	RW	0	Indicates which of the transmit endpoints (EP0, EP1, EP2, EP3, EP4) have disabled the double packet buffer functionality.
C_T_UCH_REG (0x40043344)	0x0344	16	RW	N/A	Sets the chirp timeout. This number when multiplied by 4 represents the number of XCLK cycles before the timeout occurs. That is, if XCLK is 30 MHz, this number represents the number of 133 ns time intervals before the timeout occurs. If XCLK is 60 MHz, this number represents the number of 67 ns time intervals before the timeout occurs. Although this bit is written by the host in the CLK domain, the counter that utilizes this value is in the XCLK domain. No time domain crossing is provided because the value in this register is a static.
C_T_HHSRTN_REG (0x40043346)	0x0346	16	RW	N/A	Sets the delay from the end of high speed resume signaling (acting as a Host) to enable the UTM normal operating mode. This number when multiplied by 4 represents the number of XCLK cycles before the timeout occurs. That is, if XCLK is 30 MHz, this number represents the number of 33.3 ns time intervals before the timeout occurs. If XCLK is 60 MHz, this number represents the number of 16.7 ns time intervals before the timeout occurs. Although this bit is written by the host in the CLK domain, the counter that utilizes this value is in the XCLK domain. No time domain crossing is provided because the value in this register is a static.

Table 284 • Extended Registers Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
C_T_HSBT_REG (0x40043348)	0x0348	4	RW	0	Per USB 2.0, Section 7.1.19.2, a high speed host or device expecting a response to a transmission must not timeout the transaction, if the inter packet delay is less than 736 bit times, and it must timeout the transaction if no signaling is seen within 816 bit times. Represents the value to be added to the minimum high speed timeout period of 736 bit times. The timeout period can be increased in increments of 64 high speed bit times (133 ns). There are 16 possible values. By default, the adder is 0 thus setting the high speed timeout to its minimum value. Use of this register allows the high speed timeout to be set to values that are greater than maximum specified in USB 2.0, making the USB controller non-compliant.

10.3.11.1 EPx_RQ_PKT_COUNT_REG Bit Definitions

Table 285 • EPx_RQ_PKT_COUNT_REG (0x40043XYZ)

Bit Number	Name	Reset Value	Function
[15:0]	EPx_RqPktCount	0	Sets the number of packets of size MaxP (EPx_TX_MAX_P_REG) that are to be transferred in a block transfer. Only used in Host mode when AutoReq is set. This has no effect in Peripheral mode, or when AutoReq is not set. The value of x can be 0, 1, 2, 3, or 4. The values of XYZ would be differ with the register name: 1. EP0_RQ_PKT_COUNT_REG 0x4004300 2. EP1_RQ_PKT_COUNT_REG 0x4004304 3. EP2_RQ_PKT_COUNT_REG 0x4004308 4. EP3_RQ_PKT_COUNT_REG 0x400430C 5. EP4_RQ_PKT_COUNT_REG 0x4004310

10.3.11.2 RX_DPKT_BUF_DIS_REG Bit Definitions

Table 286 • RX_DPKT_BUF_DIS_REG (0x40043340)

Bit Number	Name	Reset Value	Function
[15:5]	Reserved	N/A	
4	EP4 RxDPktBufDis	0	Receive Double Packet Buffer Disable for endpoint 4.
3	EP3 RxDPktBufDis	0	Receive Double Packet Buffer Disable for endpoint 3.
2	EP2 RxDPktBufDis	0	Receive Double Packet Buffer Disable for endpoint 2.
1	EP1 RxDPktBufDis	0	Receive Double Packet Buffer Disable for endpoint 1.
0	Reserved	N/A	

10.3.11.3 TX_DPKT_BUF_DIS_REG Bit Definitions

Table 287 • TX_DPKT_BUF_DIS_REG (0x40043342)

Bit Number	Name	Reset Value	Function
[15:5]	Reserved	N/A	
4	EP4 TxDPktBufDis	0	Transmit Double Packet Buffer Disable for endpoint 4.
3	EP3 TxDPktBufDis	0	Transmit Double Packet Buffer Disable for endpoint 3.
2	EP2 TxDPktBufDis	0	Transmit Double Packet Buffer Disable for endpoint 2.
1	EP1 TxDPktBufDis	0	Transmit Double Packet Buffer Disable for endpoint 1.
0	Reserved	N/A	

10.3.11.4 C_T_UCH_REG Bit Definitions

Table 288 • C_T_UCH_REG (0x40043344)

Bit Number	Name	Reset Value	Function
[15:0]	C_T_UCH	N/A	Configurable Chirp Timeout timer. The default value is 203Ah if the host PHY data width is 16 bits (XCLK is 30 MHz) and 4074h if the PHY data width is 8 bits (XCLK is 60 MHz), corresponding to a delay of 1.1 ms.

10.3.11.5 C_T_HHRSTN_REG Bit Definitions

Table 289 • C_T_HHSRTN_REG (0x40043346)

Bit Number	Name	Reset Value	Function
[15:0]	C_T_HHRSTN	N/A	The delay from the end of high speed resumes signaling to enabling UTM normal operating mode. The default value is 2F3h if the host PHY data width is 16 bits (XCLK is 30 MHz) and 5E6h if the PHY data width is 8 bits (XCLK is 60 MHz), corresponding to a delay of 100 μ s.

10.3.11.6 C_T_HSBT_REG Bit Definitions

Table 290 • C_T_HSBT_REG (0x40043348)

Bit Number	Name	Reset Value	Function
[3:0]	HS Timeout Adder	0	The value added to the minimum high speed timeout period (736 bit times) in increments of 64 high speed bit times. This allows the turn around timeout period to be set to 16 possible values as given in Table 291 , page 354.

10.3.11.7 Turnaround Timeout Period Settings

Table 291 • Turnaround Timeout Period Settings

Register Value	HS Turnaround Timeout (HS bit times)	HS Turnaround Timeout (μs)
0	736	1.534
1	800	1.667
2	864	1.801
3	928	1.934
4	992	2.067
5	1,056	2.201
6	1,120	2.334
7	1,184	2.467
8	1,248	2.601
9	1,312	2.734
10	1,376	2.868
11	1,140	3.001
12	1,504	3.134
13	1,568	3.268
14	1,632	3.401
15	1,696	3.534

10.3.12 Direct Memory Access (DMA) Registers

These registers correspond to the builtin DMA engine. The DMA engine has four channels. This section covers all registers in this category along with the address offset, functionality, and per bit details.

10.3.12.1 DMA_REGISTER Description

Table 292 • DMA_REGISTER Description

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
DMA_INT_REG (0x40043200)	0x0200	8	R	0	Provides an interrupt for each DMA channel. This interrupt register is cleared when read. When any bit of this register is set, the output DMA_NINT (output from DMA USB controller going to the Cortex-M3 processor) is asserted low. Bits in this register is set if the DMA Interrupt Enable bit for the corresponding channel is enabled (CHx_DMA_CNTL_REG.bit3).
CH1_DMA_CTRL_REG	0x0204	10	RW	0	Provides the DMA transfer control for channel 1. The enabling, transfer direction, transfer mode, and the DMA Burst modes are all controlled by this register.

Table 292 • DMA_REGISTER Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
CH1_DMA_ADDR_REG	0x0208	32	RW	0	Identifies the current memory address of the DMA channel 1. The initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, CH1_DMA_ADDR_ADDR[1:0] must be equal to 00. The lower two bits of this register are read only and cannot be set by the software. As the DMA transfer progresses, the memory address is incremented as bytes are transferred.
CH1_DMA_COUNT_REG	0x020C	32	RW	0	Identifies the current DMA count of the transfer for DMA channel 1. Software sets the initial count of the transfer, which identifies the entire transfer length. As the count progresses, this count is decremented as bytes are transferred.
CH2_DMA_CTRL_REG	0x0214	10	RW	0	Provides the DMA transfer control for channel 2. The enabling, transfer direction, Transfer mode, and the DMA Burst modes are controlled by this register.
CH2_DMA_ADDR_REG	0x0218	32	RW	0	Identifies the current memory address of DMA channel 2. The initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, CH2_DMA_ADDR_ADDR[1:0] must be equal to 00. The lower two bits of this register are read only and cannot be set by the software. As the DMA transfer progresses, the memory address is incremented as bytes are transferred.
CH2_DMA_COUNT_REG	0x021C	32	RW	0	Identifies the current DMA count of the transfer for DMA channel 2. The software sets the initial count of the transfer, which identifies the entire transfer length. As the count progresses, this count is decremented as bytes are transferred.
CH3_DMA_CTRL_REG	0x0224	10	RW	0	Provides the DMA transfer control for channel 3. The enabling, transfer direction, transfer mode, and the DMA Burst modes are controlled by this register.
CH3_DMA_ADDR_REG	0x0228	32	RW	0	Identifies the current memory address of DMA channel 3. The initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, CH3_DMA_ADDR_ADDR[1:0] must be equal to 00. The lower two bits of this register are only read and cannot be set by the software. As the DMA transfer progresses, the memory address is incremented as bytes are transferred.
CH3_DMA_COUNT_REG	0x022C	32	RW	0	Identifies the current DMA count of the transfer for DMA channel 3. Software sets the initial count of the transfer, which identifies the entire transfer length. As the count progresses, this count is decremented as bytes are transferred.

Table 292 • DMA_REGISTER Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
CH4_DMA_CTRL_REG	0x0234	10	RW	0	This register provides the DMA transfer control for Channel 4. The enabling, transfer direction, Transfer mode, and the DMA burst modes are controlled by this register.
CH4_DMA_ADDR_REG	0x0238	32	RW	0	Identifies the current memory address of DMA channel 4. The initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, CH4_DMA_ADDR_ADDR[1:0] must be equal to 00. The lower two bits of this register are only read and cannot be set by the software. As the DMA transfer progresses, the memory address is incremented as bytes are transferred.
CH4_DMA_COUNT_REG	0x023C	32	RW	0	Identifies the current DMA count of the transfer for DMA channel 4. Software sets the initial count of the transfer, which identifies the entire transfer length. As the count progresses, this count is decremented as bytes are transferred.

10.3.12.2 DMA_INT_REG Bit Definitions

Table 293 • DMA_INT_REG (0x40043200)

Bit Number	Name	Reset Value	Function
[7:4]	Reserved	N/A	
3	CH4_DMA_INTR	0	Channel4 DMA interrupt
2	CH3_DMA_INTR	0	Channel3 DMA interrupt
1	CH2_DMA_INTR	0	Channel2 DMA interrupt
0	CH1_DMA_INTR	0	Channel1 DMA interrupt
[7:4]	Reserved	N/A	

10.3.12.3 CHx_DMA_CTRL_REG Bit Definitions

Table 294 • CHx_DMA_CTRL_REG (0x40043204)

Bit Number	Name	Reset Value	Function
[10:9]	DMA_BRSTM	0	Burst Mode 00: Burst mode 0; bursts of unspecified length 01: Burst mode 1; INCR4 or unspecified length 10: Burst mode 2; INCR8, INCR4 or unspecified length 11: Burst mode 3; INCR16, INCR8, INCR4, or unspecified length

Table 294 • CHx_DMA_CTRL_REG (0x40043204) (continued)

Bit Number	Name	Reset Value	Function
8	DMA_ERR	0	Bus error bit. Indicates that a bus error has been observed on the input AHB_HRESPM[1:0] coming from the AHB bus matrix, originating from the Cortex-M3 processor (or fabric master). This bit is cleared by the software.
[7:4]	DMAEP	0	The endpoint number (EP0/EP1/EP2/EP3/EP4) this channel is assigned to.
3	DMAIE	0	DMA interrupt enable
2	DMAMODE	0	Selects DMA Transfer mode. 0: DMA Mode 0 transfer 1: DMA Mode 1 transfer
3	DMA_DIR	0	Selects the DMA transfer direction. 0: DMA write (receive endpoint) 1: DMA read (transmit endpoint)
2	DMA_ENAB	0	Enables the DMA transfer and will cause the transfer to begin.

Notes:

- Allowed values of x are 1, 2, 3, and 4, corresponding to DMA channels 1 through 4.
- For CH2_DMA_CTRL_REG register the address is 0x40043214.
- For CH3_DMA_CTRL_REG register the address is 0x40043224
- For CH4_DMA_CTRL_REG register the address is 0x40043234.

10.3.12.4 CHx_DMA_ADDR_REG Bit Definitions**Table 295 • CHx_DMA_ADDR_REG (0x40043208)**

Bit Number	Name	Reset Value	Function
[31:0]	DMA_ADDR	0	The DMA memory address The initial memory address written to this register must have a value such that its modulo 4 value is equal to 0. That is, DMA_ADDR[1:0] must be equal to 00. The lower two bits of this register are read only and cannot be set by software.

Notes:

- Allowed values of x are 1, 2, 3, and 4, corresponding to DMA channels 1 through 4.
- For CH2_DMA_ADDR_REG register the address is 0x40043218.
- For CH3_DMA_ADDR_REG register the address is 0x40043228.
- For CH4_DMA_ADDR_REG register the address is 0x40043238

10.3.12.5 CHx_DMA_COUNT_REG Bit Definitions

Table 296 • CHx_DMA_COUNT_REG (0x4004320C)

Bit Number	Name	Reset Value	Function
[31:0]	DMA_COUNT	0	The DMA memory address for the corresponding DMA channel. If DMA is enabled with a count of 0, the bus will not be requested and a DMA interrupt will be generated.

Notes:

- Allowed values of x are 1, 2, 3, and 4, corresponding to DMA channels 1 through 4.
- For CH2_DMA_COUNT_REG register the address is 0x4004321C.
- For CH3_DMA_COUNT_REG register the address is 0x4004322C.
- For CH4_DMA_COUNT_REG register the address is 0x4004323C.

10.3.13 Multipoint Control and Status Registers

This section covers all registers in this category along with the address offset, functionality, and per bit details.

10.3.13.1 Additional Multipoint CSR Registers

Table 297 • Additional Multipoint CSR Description

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP0_TX_FUNC_ADDR_REG	0x0080	7	RW	0	This register is used to record the address of the target function that is to be accessed through endpoint0 for transmit. Required in Host mode. For endpoint0 there is no companion EP0_RX_FUNC_ADDR_REG for receive.
EP0_TX_HUB_ADDR_REG	0x0082	8	RW	0	This register only needs to be written where a full speed or low speed device is connected to the transmit endpoint0 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full/low speed transmission. In such circumstances: <ul style="list-style-type: none"> – The lower 7 bits should record the address of this USB 2.0 hub. – The top bit should record whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only. For endpoint 0 there is no companion EP0_RX_HUB_ADDR_REG for receive.

Table 297 • Additional Multipoint CSR Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP0_TX_HUB_PORT_REG	0x0083	7	RW	0	This register only needs to be written where a full speed or low speed device is connected to the transmit endpoint0 via a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint1 is accessed. This is only relevant in Host mode. For endpoint 0 there is no companion EP0_TX_HUB_PORT_REG for receive.
EP1_TX_FUNC_ADDR_REG	0x0088	7	RW	0	This register is used to record the address of the target function that is to be accessed through endpoint1 for transmit. Required in Host mode.
EP1_TX_HUB_ADDR_REG	0x008A	8	RW	0	This register only needs to be written where a full speed or low speed device is connected to transmit endpoint1 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full/low speed transmission. In such circumstances: – The lower 7 bits should record the address of this USB 2.0 hub. – The top bit should record whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.
EP1_TX_HUB_PORT_REG	0x008B	7	RW	0	Needs to be written where a full or low speed device is connected to transmit endpoint1 through a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint0 is accessed. This is only relevant in Host mode.

Table 297 • Additional Multipoint CSR Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP1_RX_HUB_ADDR_REG	0x008E	8	RW	0	Need to be written where a full speed or low speed device is connected to receive endpoint1 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full/low speed transmission. In such circumstances: – The lower 7 bits should record the address of this USB 2.0 hub. – The top bit should record whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.
EP1_RX_HUB_PORT_REG	0x008F	7	RW	0	Needs to be written where a full speed or low speed device is connected to receive endpoint1 through a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint0 is accessed. This is only relevant in Host mode.
EP2_TX_FUNC_ADDR_REG	0x0090	7	RW	0	Records the address of the target function that is to be accessed through endpoint2 for transmit. Required in Host mode.
EP2_TX_HUB_ADDR_REG	0x0092	8	RW	0	Needs to be written where a full speed or low speed device is connected to transmit endpoint2 via a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full/low speed transmission. In such circumstances: – The lower 7 bits should record the address of this USB 2.0 hub. – The top bit should record whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.
EP2_TX_HUB_PORT_REG	0x0093	7	RW	0	Needs to be written where a full speed or low speed device is connected to transmit endpoint2 through a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with endpoint0 is accessed. This is only relevant in Host mode.

Table 297 • Additional Multipoint CSR Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP2_RX_FUNC_ADDR_REG	0x0094	7	RW	0	Records the address of the target function that is to be accessed through endpoint2 for receive. Required in Host mode.
EP2_RX_HUB_ADDR_REG	0x0096	8	RW	0	Needs to be written where a full speed or low speed device is connected to receive endpoint2 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full speed/low speed transmission. In such circumstances: – The lower 7 bits should record the address of this USB 2.0 hub. – The top bit records whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.
EP2_RX_HUB_PORT_REG	0x0097	7	RW	0	Needs to be written where a full speed or low speed device is connected to receive endpoint2 via a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint0 is accessed. This is only relevant in Host mode.
EP3_TX_FUNC_ADDR_REG	0x0098	7	RW	0	Records the address of the target function that is to be accessed through endpoint3 for transmit. Required in Host mode.
EP3_TX_HUB_ADDR_REG	0x009A	8	RW	0	Needs to be written where a full speed or low speed device is connected to transmit endpoint3 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full speed/low speed transmission. In such circumstances: – The lower 7 bits records the address of this USB 2.0 hub. – The top bit records whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.

Table 297 • Additional Multipoint CSR Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP3_TX_HUB_PORT_REG	0x009B	7	RW	0	Needs to be written where a full speed or low speed device is connected to transmit endpoint3 through a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint0 is accessed. This is only relevant in Host mode.
EP3_RX_FUNC_ADDR_REG	0x009C	7	RW	0	Records the address of the target function that is to be accessed through endpoint3 for receive. Required in Host mode.
EP3_RX_HUB_ADDR_REG	0x009E	8	RW	0	Needs to be written where a full speed or low speed device is connected to receive endpoint3 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full/low speed transmission. In such circumstances: – The lower 7 bits records the address of this USB 2.0 hub. – The top bit records whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.
EP3_RX_HUB_PORT_REG	0x009F	7	RW	0	Needs to be written where a full speed or low speed device is connected to receive endpoint3 through a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint0 is accessed. This is relevant only in Host mode.
EP4_TX_FUNC_ADDR_REG	0x00A0	7	RW	0	Records the address of the target function that is to be accessed through endpoint4 for transmit. Required in Host mode.

Table 297 • Additional Multipoint CSR Description (continued)

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
EP4_TX_HUB_ADDR_REG	0x00A2	8	RW	0	Needs to be written where a full speed or low speed device is connected to transmit endpoint4 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full/low speed transmission. In such circumstances: – The lower 7 bits records the address of this USB 2.0 hub. – The top bit records whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.
EP4_TX_HUB_PORT_REG	0x00A3	7	RW	0	Needs to be written where a full speed or low speed device is connected to transmit endpoint4 through a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint0 is accessed. This is only relevant in Host mode.
EP4_RX_FUNC_ADDR_REG	0x00A4	7	RW	0	Records the address of the target function that is to be accessed through endpoint4 for receive. Required in Host mode.
EP4_RX_HUB_ADDR_REG	0x00A6	8	RW	0	Needs to be written where a full or low speed device is connected to receive endpoint4 through a high speed USB 2.0 hub which carries out the necessary transaction translation to convert between high speed transmission and full/low speed transmission. In such circumstances: – The lower 7 bits records the address of this USB 2.0 hub. – The top bit records whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators). This is relevant in Host mode only.
EP4_RX_HUB_PORT_REG	0x00A7	7	RW	0	Needs to be written where a full speed or low speed device is connected to receive endpoint4 through a high speed USB 2.0 hub which carries out the necessary transaction translation. In such circumstances, these 7-bit read/write registers need to be used to record the port of that USB 2.0 hub through which the target associated with the endpoint0 is accessed. This is only relevant in Host mode.

10.3.13.2 EPx_TX_FUNC_ADDR_REG Bit Definitions

Table 298 • EPx_TX_FUNC_ADDR_REG (0x40043080)

Bit Number	Name	Reset Value	Function
[6:0]	TxFuncAddr	0	Address of target function for transmit endpointx

Notes:

- Allowed values of x are 0, 1, 2, 3, and 4, corresponding to endpoints 0, 1, 2, 3, and 4.
- For EP1_TX_FUNC_ADDR_REG register the address is 0x40043088.
- For EP2_TX_FUNC_ADDR_REG register the address is 0x40043090.
- For EP3_TX_FUNC_ADDR_REG register the address is 0x40043098.
- For EP4_TX_FUNC_ADDR_REG register the address is 0x400430A0.

10.3.13.3 EPx_RX_FUNC_ADDR_REG Bit Definitions

Table 299 • EPx_RX_FUNC_ADDR_REG

Bit Number	Name	Reset Value	Function
[6:0]	RxFuncAddr	0	Address of target function for Receive Endpointx

Notes:

- Allowed values of x are 0, 1, 2, 3, and 4, corresponding to endpoints 0, 1, 2, 3, and 4.
- For EP2_RX_FUNC_ADDR_REG register the address is 0x40043094.
- For EP3_RX_FUNC_ADDR_REG register the address is 0x4004309C.
- For EP4_RX_FUNC_ADDR_REG register the address is 0x400430A4.

10.3.13.4 EPx_TX_HUB_ADDR_REG Bit Definitions

Table 300 • EPx_TX_HUB_ADDR_REG (0x40043082)

Bit Number	Name	Reset Value	Function
7	Multiple Translators	0	Records whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators).
[6:0]	Tx Hub Address	0	Address of this USB 2.0 hub is used for transmit

Notes:

- Allowed values of x are 0, 1, 2, 3, and 4, corresponding to endpoints 0, 1, 2, 3, and 4.
- For EP1_TX_HUB_ADDR_REG register the address is 0x4004308A.
- For EP2_TX_HUB_ADDR_REG register the address is 0x40043092.
- For EP3_TX_HUB_ADDR_REG register the address is 0x4004309A.
- For EP4_TX_HUB_ADDR_REG register the address is 0x400430A2.

10.3.13.5 EPx_RX_HUB_ADDR_REG Bit Definitions

Table 301 • EPx_RX_HUB_ADDR_REG

Bit Number	Name	Reset Value	Function
7	Multiple Translators	0	Records whether the hub has multiple transaction translators (set to '0' if single transaction translator; set to '1' if multiple transaction translators).
[6:0]	Rx Hub Address	0	Address of this USB 2.0 hub for receive

Notes:

- Allowed values of x are 0, 1, 2, 3, and 4, corresponding to endpoints 0, 1, 2, 3, and 4.
- For EP1_RX_HUB_ADDR_REG register the address is 0x4004308E.
- For EP2_RX_HUB_ADDR_REG register the address is 0x40043096.
- For EP3_RX_HUB_ADDR_REG register the address is 0x4004309E.
- For EP4_RX_HUB_ADDR_REG register the address is 0x400430A6.

10.3.13.6 EPx_TX_HUB_PORT_REG Bit Definitions

Table 302 • EPx_TX_HUB_PORT_REG (0x40043083)

Bit Number	Name	Reset Value	Function
[6:0]	TxHubPort	0	This information, together with the hub address (EPx_TX_HUB_ADDR_REG), allows the USB controller to support split transactions.

Notes:

- For EP1_TX_HUB_PORT_REG register the address is 0x4004308B.
- For EP2_TX_HUB_PORT_REG register the address is 0x40043093.
- For EP3_TX_HUB_PORT_REG register the address is 0x4004309B.
- For EP4_TX_HUB_PORT_REG register the address is 0x400430A3.

10.3.13.7 EPx_RX_HUB_PORT_REG Bit Definitions

Table 303 • EPx_RX_HUB_PORT_REG

Bit Number	Name	Reset Value	Function
[6:0]	RxHubPort	0	This information, together with the hub address (EPx_RX_HUB_ADDR_REG), allows the USB controller to support split transactions.

Notes:

- For EP1_RX_HUB_PORT_REG register the address is 0x4004308F.
- For EP2_RX_HUB_PORT_REG register the address is 0x40043097.
- For EP3_RX_HUB_PORT_REG register the address is 0x4004309F.
- For EP4_RX_HUB_PORT_REG register the address is 0x400430A7.

10.3.14 Link Power Management Registers

10.3.14.1 Link Power Management Register Descriptions

Table 304 • Link Power Management Register Descriptions

Register Name	Address Offset from 0x40043000	Width	R/W Type	Reset Value	Description
LPM_ATTR_REG (0x40043360)	0x0360	16	R	0	Defines the attributes of an LPM transaction and sleep cycle. In both Host mode and Peripheral mode, the meaning of this register is same; however, the source of the data is different for Host and Peripheral modes as follows: In Peripheral mode: The values in this register contains the equivalent attributes that were received in the last LPM transaction that was accepted. This register is updated with the LPM packet contents if the response to the LPM transaction was an ACK. This register can be updated through software. In all other cases, this register holds its current value. In Host mode: Software sets up the values in this register to define the next LPM transaction that is transmitted. These values are inserted in the payload of the next LPM transaction.
LPM_CTRL_REG (0x40043362)(Peripheral) LPM_CTRL_REG (0x40043362) (Host)	0x0362	8	R	0	Provides controls for LPM based on Peripheral mode and Host mode.
LPM_INTR_EN_REG (0x40043363)	0x0363	8	R	0	Provides enable bits for the interrupts in LPM_INTR_REG. If a bit in this register is set to 1, MC_NINT will be asserted (low) when the corresponding interrupt in the LPM_INTR_REG is set. If a bit in this register is set to 0, the corresponding register in LPM_INTR_REG is still set but MC_NINT will not be asserted (low). On reset, all bits in this register are reset to 0.
LPM_INTR_REG (0x40043364)(Peripheral Mode) LPM_INTR_REG (0x40043364) (Host Mode)	0x0364	7	R	0	Provides status of the LPM power state. When a bit is set to 1, if the corresponding enable bit is also set to 1, the output MC_NINT is asserted (low). If the corresponding enable bit is set to 0, then the output MC_NINT is not asserted. On reset, all bits in this register are reset to 0. This register is clear on read.
LPM_FADDR_REG (0x40043365)	0x0365	7	R	0	Holds the function address that is placed in the LPM payload. This has relevance in Host mode only.

10.3.14.2 LPM_ATTR_REG Bit Definitions

Table 305 • LPM_ATTR_REG (0x40043360)

Bit Number	Name	Reset Value	Function
[15:12]	EndPnt	0	This is the endpoint that is in the token packet of the LPM transaction.
[11:9]	Reserved	N/A	
8	RmtWak	0	This bit is the remote wake-up enable bit. 0: Remote wake-up is not enabled. 1: Remote wake-up is enabled This bit is applied on a temporary basis and is applied only to the current suspend state. After the current suspend cycle, the remote wake-up capability that was negotiated upon enumeration applies.
[7:4]	HIRD	0	This is the host initiated resume duration. This value is the minimum time the host drives resume on the bus. The value in this register corresponds to an actual resume time as follows: Resume Time = 50 μ s + HIRD \times 75 μ s. This results a range 50 μ s to 1200 μ s.
[3:0]	LinkState	0	This value is provided by the host to the peripheral to indicate what state the peripheral must transition to after the receipt and acceptance of a LPM transaction. 0000: Reserved 0001: Sleep state (L1) 0010: Reserved 0011: Reserved

10.3.14.3 LPM_CTRL_REG (Peripheral) Bit Definitions

Table 306 • LPM_CTRL_REG (0x40043362)(Peripheral)

Bit Number	Name	Reset Value	Function
[7:5]	Reserved	N/A	
4	LPMNAK	0	Places all endpoints in a state such that the response to all transactions other than an LPM transaction is a NAK. This bit takes effect only after the USB controller has been LPM suspended. In this case, the USB controller continues to NAK until this bit has been cleared by software.
[3:2]	LPMEN	0	Enables LPM in the USB controller. There are three levels in which LPM can be enabled, which determines the response of the USB controller to LPM transactions. Following are the three levels: 00: LPM and extended transactions are not supported. In this case, the USB controller does not respond to LPM transactions and the transaction timeouts. 01: LPM and extended transactions are not supported. In this case, the USB controller does not respond to LPM transactions and the transaction timeouts. 10: LPM is not supported but extended transactions are supported. In this case, the USB controller responds to an LPM transaction with a STALL. 11: The USB controller supports LPM extended transactions. In this case, the USB controller responds with a NYET or an ACK as determined by the value of LPMXMT (bit 0 of this register) and other conditions.
1	LPMRES	0	Initiates resume (remote wake-up). This bit differs from the classic RESUME bit (POWER_REG.bit2) in that the RESUME signal timing is controlled by hardware. When software writes this bit, resume signaling is asserted for 50 μ s. This bit is self clearing.

Table 306 • LPM_CTRL_REG (0x40043362)(Peripheral) (continued)

Bit Number	Name	Reset Value	Function
0	LPMXMT	0	<p>Instructs the USB controller to transition to the L1 state upon the receipt of the next LPM transaction. This bit is only effective if LPMEN (bits[3:2] of this register) is set to 11. This bit can be set in the same cycle as LPMEN. If this bit is set to 1 and LPMEN = 11, the USB controller can respond in the following ways:</p> <p>If any data is not pending (all transmit FIFOs are empty), the USB controller will respond with an ACK. In this case this bit will self clear and a software interrupt will be generated.</p> <p>If any data is pending (data resides in at least one transmit FIFO), the USB controller will respond with a NYET. In this case, this bit will NOT self clear; however, a software interrupt will be generated.</p>

10.3.14.4 LPM_CTRL_REG (Host) Bit Definitions

Table 307 • LPM_CTRL_REG (0x40043362) (Host)

Bit Number	Name	Reset Value	Function
[7:2]	Reserved	N/A	
1	LPMRES	0	<p>Initiates a RESUME from the L1 state. This bit differs from the classic RESUME bit (POWER_REG.bit2) in that the RESUME signal timing is controlled by hardware. When software writes this bit, resume signaling is asserted for a time specified by the HIRD field (LPM_ATTR_REG.bit[7:4]). This bit is self clearing.</p>
0	LPMXMT	0	<p>Transmits an LPM transaction. This bit is self clearing. This bit will be immediately cleared upon receipt of any token or three timeouts have occurred.</p>

10.3.14.5 LMP_INTR_EN_REG Bit Definitions

Table 308 • LPM_INTR_EN_REG (0x40043363)

Bit Number	Name	Reset Value	Function
[7:6]	Reserved	N/A	
5	LPMERREN	0	0: Disable the LPMERR interrupt 1: Enable the LPMERR interrupt
4	LPMRESEN	0	0: Disable the LPMRES interrupt 1: Enable the LPMRES interrupt
3	LPMNCEN	0	0: Disable the LPMNC interrupt 1: Enable the LPMNC interrupt
2	LPMACKEN	0	0: Disable the LPMACK interrupt 1: Enable the LPMACK interrupt
1	LPMNYEN	0	0: Disable the LPMNY interrupt 1: Enable the LPMNY interrupt
0	LPMSTEN	0	0: Disable the LPMST interrupt 1: Enable the LPMST interrupt

10.3.14.6 LPM_INTR_REG (Peripheral) Bit Definitions

Table 309 • LPM_INTR_REG (0x40043364)(Peripheral Mode)

Bit Number	Name	Reset Value	Function
[7:6]	Reserved	N/A	
5	LPMERR	0	This bit is set if an LPM transaction is received that has a LinkState field (LINK_ATTR_REG.bits[3:0]) that is not supported. In this case, the response to the transaction is a STALL. However, the LINK_ATTR_REG is updated so that software can observe the noncompliant LPM packet payload.
4	LPMRES	0	This bit is set when the USB controller has been resumed for any reason. This bit is mutually exclusive from RESUME (POWER_REG.bit2).
3	LPMNC	0	This bit is set when an LPM transaction is received and the USB controller responds with a NYET due to data pending in the received FIFOs. This can only occur under the following condition: The LPMEN field (LPM_CTRL_REG.bit[3:2]) is set to 11, the LPMXMT (LPM_CTRL_REG.bit0) is set to 1, and there is data pending in the USB controller transmit FIFOs.
2	LPMACK	0	This bit is set when an LPM transaction is received and the USB controller responds with an ACK. This can only occur under the following condition: The LPMEN field (LPM_CTRL_REG.bit[3:2]) is set to 11, the LPMXMT (LPM_CTRL_REG.bit0) is set to 1, and there is data pending in the USB controller transmit FIFOs.
1	LPMNY	0	This bit is set when an LPM transaction is received and the USB controller responds with a NYET. This can only occur under the following conditions: The LPMEN field (LPM_CTRL_REG.bit[3:2]) is set to 11, the LPMXMT (LPM_CTRL_REG.bit0) is set to 1, and there is data pending in the USB controller transmit FIFOs.
0	LPMST	0	This bit is set when an LPM transaction is received and the USB controller responds with a STALL. This can only occur under the following condition: The LPMEN field (LPM_CTRL_REG.bit[3:2]) is set to 01.

10.3.14.7 LPM_INTR_REG (Host) Bit Definitions

Table 310 • LPM_INTR_REG (0x40043364) (Host Mode)

Bit Number	Name	Reset Value	Function
[7:6]	Reserved	N/A	
5	LPMERR	0	This bit is set if a response to the LPM transaction is received with a bit stuff error or a PID error. In this case, suspend does not occur and the state of the device is unknown.
4	LPMRES	0	This bit is set when the USB controller has been resumed for any reason. This bit is mutually exclusive from RESUME (POWER_REG.bit2).
3	LPMNC	0	This bit is set when an LPM transaction has been transmitted and has failed to complete. The transaction would have failed either because a timeout is occurred or there are bit errors in the response for three attempts.
2	LPMACK	0	This bit is set when an LPM transaction is transmitted and the device responds with an ACK.

Table 310 • LPM_INTR_REG (0x40043364) (Host Mode) (continued)

Bit Number	Name	Reset Value	Function
1	LPMNY	0	This bit is set when an LPM transaction is transmitted and the device responds with a NYET.
0	LPMST	0	This bit is set when an LPM transaction is transmitted and the device responds with a STALL.

10.3.14.8 LPM_FADDR_REG Bit Definitions

Table 311 • LPM_FADDR_REG (0x40043365)

Bit Number	Name	Reset Value	Function
7	Reserved	N/A	
[6:0]	LPMFADDR	0	The LPM function address

10.3.15 USB System Registers

These registers are in the System registers block for the SmartFusion2 devices. This subset of registers, and in some cases bit fields, control the configuration and behavior of the USB controller.

10.3.15.1 MASTER_WEIGHT1_CR Register USB Related Bit Definitions

Table 312 • MASTER_WEIGHT1_CR

Bit Number	Name	Address	Reset Value	Function
[9:5]	SW_WEIGHT_USB	0x40038040	0x1	Configures the round robin weightage for the USB master from 1 to 32. The configurable weightage values are as follows: 00000: USB master weightage 32 00001: USB master weightage 1 00010: USB master weightage 2 00011: USB master weightage 3 00100: USB master weightage 4 00101: USB master weightage 5 00110: USB master weightage 6 00111: USB master weightage 7 01000: USB master weightage 8 01001: USB master weightage 9 01010: USB master weightage 10 01011: USB master weightage 11 01100: USB master weightage 12 01101: USB master weightage 13 01110: USB master weightage 14 01111: USB master weightage 15 10000: USB master weightage 16 10001: USB master weightage 17 10010: USB master weightage 18 10011: USB master weightage 19 10100: USB master weightage 20 10101: USB master weightage 21 10110: USB master weightage 22 10111: USB master weightage 23 11000: USB master weightage 24 11001: USB master weightage 25 11010: USB master weightage 26 11011: USB master weightage 27 11100: USB master weightage 28 11101: USB master weightage 29 11110: USB master weightage 30 11111: USB master weightage 31

10.3.15.2 USB_IO_INPUT_SEL_CR Register Bit Definitions

Table 313 • USB_IO_INPUT_SEL_CR

Bit Number	Name	Address	Reset Value	Function
2	USB_IO_INPUT_SEL	0x40038064	0	Selects one of the four USB data interfaces from IOMUXCELLs and I/O pads. Following are the allowed values: 00: USB A interface can be connected to USB controller 01: USB B interface can be connected to USB controller 10: USB C interface can be connected to USB controller 11: USB D interface can be connected to USB controller

10.3.15.3 USB_CR Register Bit Definitions

Table 314 • USB_CR

Bit Number	Name	Address	Reset Value	Function
0	USB_UTMI_SEL	0x4003807C	0	This signal is used to configure USB controller interface as ULPI PHY or UTMI interface. Following are the allowed values: 0: ULPI PHY Interface is selected 1: UTMI Interface is selected
1	USB_DDR_SELECT		0	This signal is used to configure whether the USB controller works in Single Data Rate (SDR) mode or Double Data Rate (DDR) mode. Allowed values: 0: SDR mode is selected 1: DDR mode is selected

10.3.15.4 USB_EDAC_CNT Register Bit Definitions

Table 315 • USB_EDAC_CNT

Bit Number	Name	Address	Reset Value	Function
[15:0]	USB_EDAC_CNT_1E	0x40038104	0	This is a 16-bit counter value in USB incremented by USB EDAC 1-bit errors. The counter does not roll back and stays at its maximum value, if reached.
[31:16]	USB_EDAC_CNT_2E		0	This is a 16-bit counter value in USB incremented by USB EDAC 2-bit errors. The counter does not roll back and stays at its maximum value, if reached.

10.3.15.5 USB_EDAC_ADR Register Bit Definitions

Table 316 • USB_EDAC_ADR

Bit Number	Name	Address	Reset Value	Function
[10:0]	USB_EDAC_1E_AD	0x40038120	0	Address from USB memory on which a 1-bit SECDED error is occurred
[21:11]	USB_EDAC_2E_AD		0	Address from USB memory on which a 2-bit SECDED error is occurred

10.3.15.6 USB_SR Register Bit Definitions

Table 317 • USB_SR

Bit Number	Name	Address	Reset Value	Function
0	POWERDN	0x40038154	0	Asserted when CLK may be stopped to save power. Derived from combination of signals from CLK and XCLK flip-flops, AVALID, VBUSVALID,, and LINESTATE.)
1	LPI_CARKIT_EN		0	Asserted when entry is made into CarKit mode and cleared on exit from CarKit mode.

10.3.15.7 EDAC_SR Register USB Related Bit Definitions

Table 318 • EDAC_SR

Bit Number	Name	Address	Reset Value	Function
10	USB_EDAC_1E	0x40038190	0	This status is updated by USB when a 1-bit SECDED error is detected and also corrected for RAM memory.
11	USB_EDAC_2E		0	This status is updated by USB when a 2-bit SECDED error is detected and also corrected for RAM memory.

10.3.15.8 CLR_EDAC_COUNTERS Register USB Related Bit Definitions

Table 319 • CLR_EDAC_COUNTERS

Bit Number	Name	Address	Reset Value	Function
10	USB_EDAC_CNTCLR_1E	0x400381A4	0	This is pulse generated to clear the 16-bit counter value in USB corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of USB_EDAC_CNT register.
11	USB_EDAC_CNTCLR_2E		0	This is pulse generated to clear the 16-bit counter value in USB corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the USB_EDAC_CNT register.

11 Ethernet MAC

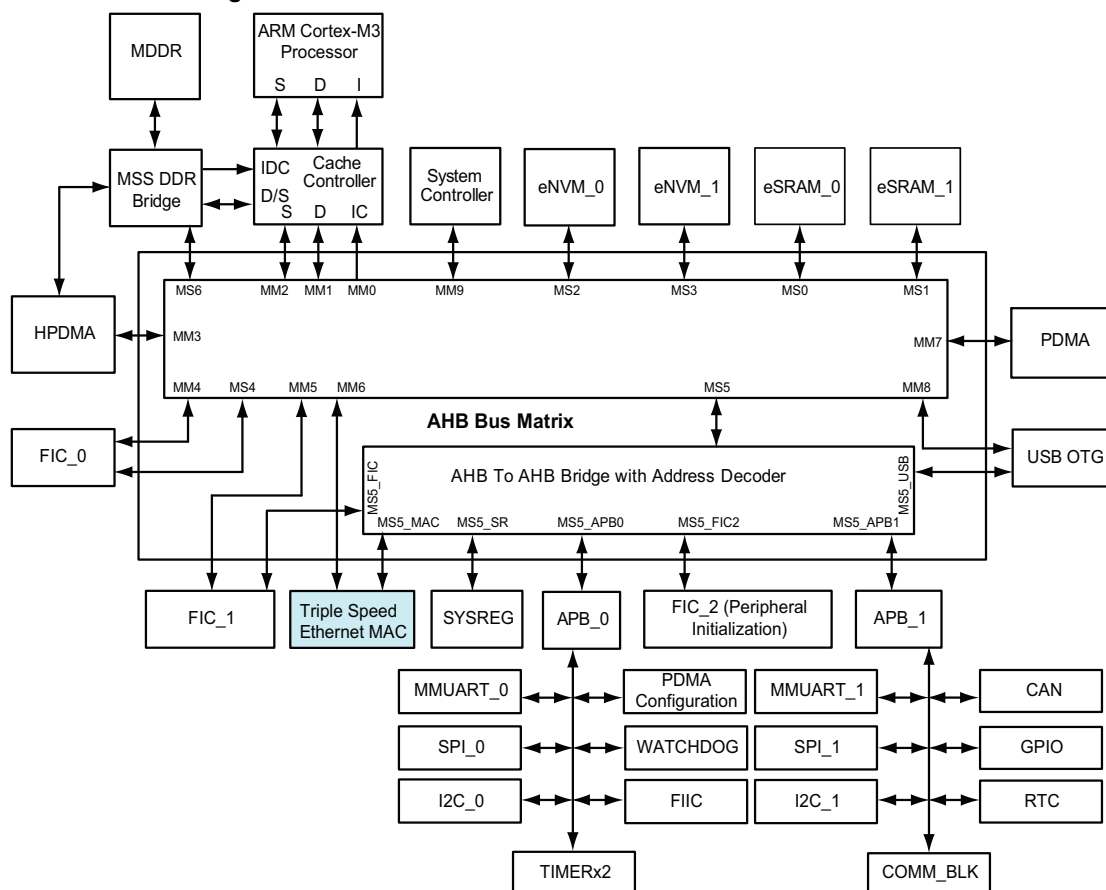
The tri-speed (10/100/1000 Mbps) Ethernet medium access controller (TSEMAC) is a medium access controller which can be configured to 10/100/1000 Mbps data transfer rate (line speed) between the host processor and Ethernet network. The SmartFusion2 SoC FPGA device has one instance of the TSEMAC peripheral as part of the microcontroller subsystem (MSS).

11.1 Features

- Supports tri-speeds: 10/100/1000 Mbps
- Implements the carrier sense multiple access with the collision detection (CSMA/CD) algorithms defined by the institute of electrical and electronics engineers (IEEE) 802.3 standard.
- The advanced high-performance bus (AHB) master port for the direct memory access (DMA) transfers and AHB-slave port for the configuration space access.
- The media independent interface (MII), gigabit media independent interface (GMII) and the ten-bit interface (TBI) for external PHY support.
- MII/GMII/TBI loopback support
- 4 KB of TX buffer and 8 KB of RX buffer
- Both TX and RX Buffers are protected by single error correction and dual error detection (SECDED)
- Standard Ethernet frames of 1522 bytes are supported. Jumbo frames of 9000 bytes are not supported.

The following figure shows the details of MSS. As shown, TSEMAC can function as an AHB master for DMA data transfers and as an AHB slave for configuring the TSEMAC from the master ARM® Cortex® -M3 processor or from the field programmable gate array (FPGA) fabric logic.

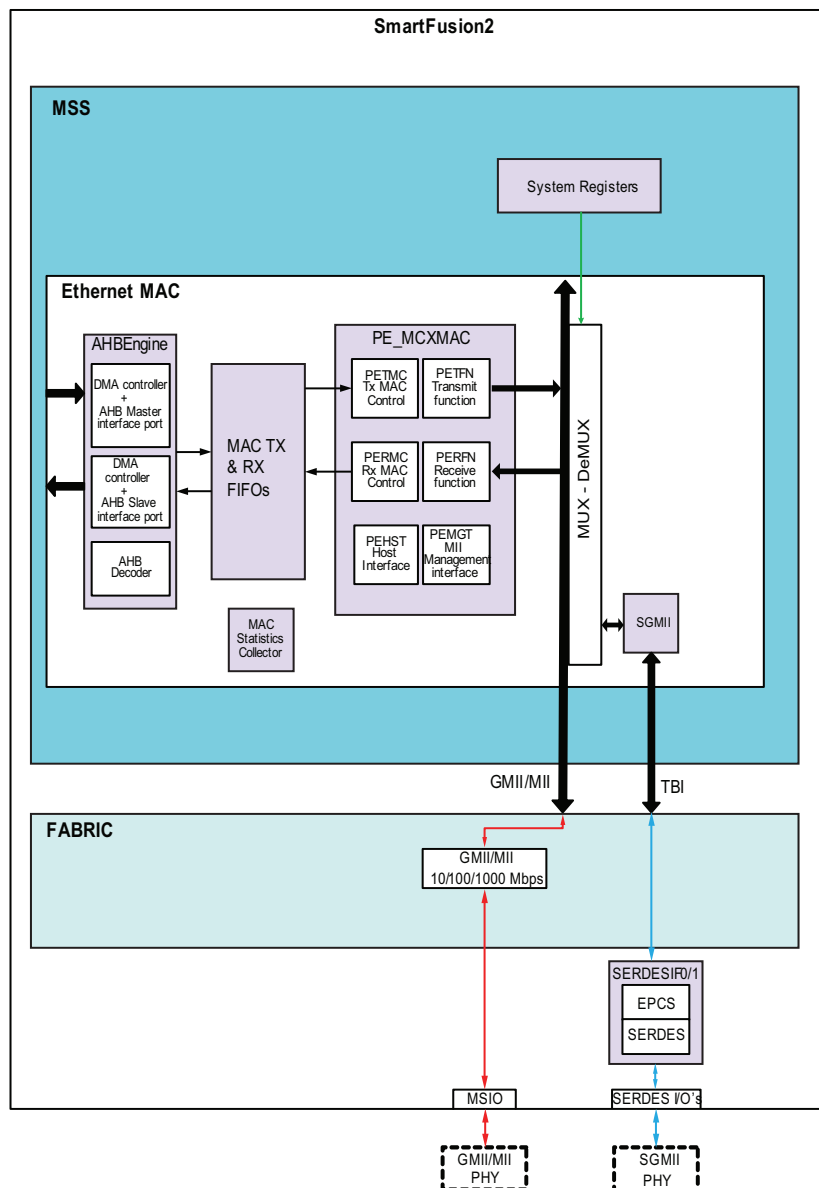
Figure 152 • MSS Showing a TSEMAC



11.2 Functional Description

The TSEMAC controller is interfaced through the advanced high-performance bus (AHB) matrix in the MSS. SmartFusion2 TSEMAC provides three interfaces (MII, GMII, and TBI) to connect to the external PHY. The following figure shows the block diagram for the connections between the EMAC and FPGA fabric.

Figure 153 • TSEMAC Block Diagram



11.2.1 EMAC Functional Blocks

EMAC has five functional sub-blocks:

- AHB Engine
- MAC TX and RX FIFO
- PE-MCXMAC
- MAC Statistics Module
- SGMII Module

11.2.1.1 AHB Engine

The EMAC can be accessed from an AHB system bus using the AHB engine. The AHB engine module is positioned between the AHB system bus and the MAC TX and RX FIFO.

The AHB Engine includes the following modules:

- **DMA:** Includes logic for the AHB master interface and contains the DMA controller.
- **Slave:** Includes logic for the AHB slave interface.
- **Decoder:** Address bus decoder module to divide accesses between MAC core, FIFO and accesses to the registers within AHB engine.

The AHB engine includes a DMA controller which is used to transmit and receive operations. Both operations compete for the use of the DMA controller. A round-robin priority algorithm is used to arbitrate between the competing requests.

The AHB engine module interfaces with the host system through a 32-bit AHB master and slave ports. The AHB engine module is positioned between the AHB system bus and the FIFO. Registers within the AHB engine provide control and status information concerning these transfers.

11.2.1.2 MAC TX and RX FIFO

The A-MCXFIFO is a flexible FIFO module with the transmit buffering and the receive buffering, which can significantly improve the performance of the embedded 10/100/1000 Mbps Ethernet systems. This FIFO module offers increased system level throughput by allowing data queueing. The size of the transmit buffer is 4 KB and the receive buffer is 8 KB.

Both automatic pause frame handshaking and per transmit frame MAC configuration data are supported. A PAUSE frame is used for flow control. This halts the transmission for a specified period of time. A PAUSE frame includes the period of pause time being requested.

Register definitions for the FIFO RAM access registers are intended for non-real-time RAM testing and system debug. MAC TX and RX FIFO configuration registers one through five are intended to be written while the submodules are held in reset. FIFO sizes are fixed and cannot be modified by either the MAC configuration or the firmware.

11.2.1.3 PE-MCXMAC

The PE-MCXMAC provides a 10/100/1000 Mbps EMAC with a GMII interface. The PE-MCXMAC supports control frames, particularly PAUSE frames. Other types of control frames can be supported through the setting optional configuration bits.

The PE-MCXMAC module consists of the following six sub-modules.

11.2.1.3.1 PETFN Transmit Function

The PETFN sub-module accepts the transmit frames, and prepends a 7-byte preamble and start of frame delimiter. The PETFN waits for the pre-programmed inter-packet gap before outputting the preamble.

11.2.1.3.2 PETMC Transmit MAC Control

The PETMC sub-module is responsible for the multiplexing of normal transmit frames and control frame requests. It provides native support to the PAUSE flow control frames.

11.2.1.3.3 PERFN Receive Function

The PERFN sub-module accepts receive packets through the GMII, extracts frames, strips off the preamble, and starts the frame delimiter from each frame before presenting them to the system. It also calculates the cyclic redundancy check (CRC) of the received frame for checking against the Frame Check Sequence field.

11.2.1.3.4 PERMC Receive MAC Control

The PERMC sub-module is responsible for detecting the control frames in the receive data stream. Each frame is examined to determine if it is a control frame and if so, whether it is a PAUSE frame.

11.2.1.3.5 PEHST Host Interface

This 32-bit interface gives access to the status and control registers included in the PE-MCXMAC.

11.2.1.3.6 PEMGT MII Management

The PEMGT sub-module drives the MII Management Interface where the control and status information is exchanged with the attached PHY.

11.2.1.4 MAC Statistics Module

The MAC Statistics module is a register-based, statistics-gathering module. The MAC Statistics module offers 37 separate counters, which can be used either to count or to accumulate conditions (such as dropped frames which occur as packets), that are transmitted or received. The presence of non-zero elements in either of these statistics vectors triggers MAC Statistics module to update its statistics counters.

The counter's rollover condition is indicated through a carry interrupt output from the MAC Statistics module. Each counters rollover condition can be discreetly masked from causing an interrupt by internal masking registers.

[Table 423](#), page 426 and [Table 424](#), page 427 show the mask registers for transmit and receive.

11.2.1.5 SGMII Module

The SGMII module provides an SGMII that facilitates a connection between any IEEE 802.3 standard GMII or MII interface and an SGMII interface that is compliant with version 1.7 of the SGMII specification.

The SGMII module uses TBI to provide the 8B/10B encoding, decoding, and auto-negotiation functionality that is defined in Clause 36 of the IEEE 802.3z specification. This allows link partners to synchronize with each other and exchange information regarding their configuration capabilities, using a symbol stream that has a proven industry standard. The SGMII also instantiates transmit and receive conversion and rate adaptation modules, which allow for G/MII data/control conversion, half-duplex control encoding, and clock domain interfacing to the SGMII clock domain. The SGMII core includes optional modules for 10-bit comma alignment.

Both transmit and receive paths leverage the physical coding sub-layer and the auto-negotiation sub-layers of the IEEE 802.3z specification, as explained in clauses 36 and 37. In the transmit direction, the 10-bit encoded data is serialized and output over the interface. In the receive direction, the single-bit input is made parallel after being aligned to comma characters.

The SGMII core also includes support for MAC speeds less than gigabit rates. To maintain a constant clock frequency at the PHY interface for all MAC speeds, the MII bus data is replicated internally to maintain a gigabit rate at the output of the SGMII core.

TSEMAC can have separate clocks for MSS and PHY side connections. These clocks are asynchronous to each other.

11.3 TSEMAC PHY Interfaces

This section describes the MII, GMII, and TBI fabric PHY interfaces. The PHY interfaces (MII, GMII, and TBI) are routed to the fabric onto MSIOs in case of MII, GMII, and SERDES I/Os in case of TBI. To implement the other PHY protocols within the fabric such as RMII, RGMII, RTBI, RevMII, and SMII, these PHY protocols may be derived from appropriate wrappers implemented in fabric, which converts PHY interface signals input to the FPGA fabric as shown in [Figure 152 on page 374](#). The EMAC can be configured through Libero SoC MSS configurator by selecting the interface, line speed, and enabling the management interface PHY interface. On reset, MSS Ethernet MAC turns the auto-negotiate of the Ethernet PHY to OFF. Due to difference in protocol between MSS Ethernet MAC and Ethernet PHY, user has to handle the Ethernet PHY auto negotiation through PHY driver code. The SGMII can be implemented by means of configuring the EMAC for the TBI operation. TBI pins are routed to SERDES, which is configured for EPCS mode to provide SGMII functionality. as shown in [Figure 153 on page 375](#).

The following tables list the port names, port groups, and direction information for the default supported MII, GMII and TBI PHY interfaces.

Table 320 • MII Ports

Port Name	Port Group	Direction	Description
MII_TXD[3:0]	MAC_MII_FABRIC	Out	Indicates MII transmit data
MII_TX_EN	MAC_MII_FABRIC	Out	Indicates MII transmit data enable
MII_TX_ER	MAC_MII_FABRIC	Out	Indicates MII transmit data error
MII_RXD[3:0]	MAC_MII_FABRIC	In	Indicates MII receive data
MII_RX_ER	MAC_MII_FABRIC	In	Indicates MII receive data error
MII_RX_DV	MAC_MII_FABRIC	In	Indicates MII receive data valid
MII_CRS	MAC_MII_FABRIC	In	Asynchronous carrier sense signal. Indicates at least one physical device transmits on the medium.
MII_COL	MAC_MII_FABRIC	In	Asynchronous collision sense signal. Indicates more than one physical device transmits simultaneously on the medium.
MII_RX_CLK	MAC_MII_FABRIC	In	Indicates MII receive clock. 25 MHz for 100 Mbps mode and 2.5 MHz for 10 Mbps mode.
MII_TX_CLK	MAC_MII_FABRIC	In	Indicates MII management transmit clock. 25 MHz for 100-Mbps mode and 2.5 MHz for 10 Mbps mode. MII_TXD, MII_TX_EN, MII_TX_ER signals are synchronized to MII_TX_CLK.
MII_MDC		Out	Indicates MII management data clock
MII_MDO_ED		Out	Indicates MII management data output enable
MII_MDO		Out	Indicates MII management data out
MII_MDI		In	Indicates MII management data input

Table 321 • GMII Ports

Port Name	Port Group	Direction	Description
GMII_TXD[7:0]	MAC_GMII_FABRIC	Out	GMII transmit data
GMII_TX_EN	MAC_GMII_FABRIC	Out	GMII transmit data enable
GMII_TX_ER	MAC_GMII_FABRIC	Out	GMII transmit data error
GMII_RXD[7:0]	MAC_GMII_FABRIC	Out	GMII receive data
GMII_RX_ER	MAC_GMII_FABRIC	In	GMII receive data error
GMII_RX_DV	MAC_GMII_FABRIC	In	Indicates MII receive data valid
GMII_CRS	MAC_GMII_FABRIC	In	Asynchronous carrier sense signal. Indicates at least one physical device transmits on the medium.
GMII_COL	MAC_GMII_FABRIC	In	Asynchronous collision signal. Indicates more than one physical device transmits simultaneously on the medium.
GMII_RX_CLK	MAC_GMII_FABRIC	In	Indicates GMII receive clock. 125 MHz for 1000-megabit mode, 25 MHz for 100-megabit mode, and 2.5 MHz for 10-megabit mode.

Table 321 • GMII Ports (continued)

Port Name	Port Group	Direction	Description
GMII_TX_CLK	MAC_GMII_FABRIC	In	Indicates GMII transmit clock. 25 MHz for 100-megabit mode and 2.5-megabit for 10-megabit mode. GMII_TXD, GMII_TX_EN, GMII_TX_ER signals are synchronized to GMII_TX_CLK.
GMII_GTX_CLK	MAC_GMII_FABRIC	In	Indicates gigabit 125 MHz transmit clock input for 1000-megabit mode. GMII_TXD, GMII_TX_EN, GMII_TX_ER signals are synchronized to GMII_GTX_CLK.
GMII_MDC		Out	GMII management data clock
GMII_MDO_EN		Out	GMII management data output enable
GMII_MDO		Out	GMII management data out
GMII_MDI		In	GMII management data input

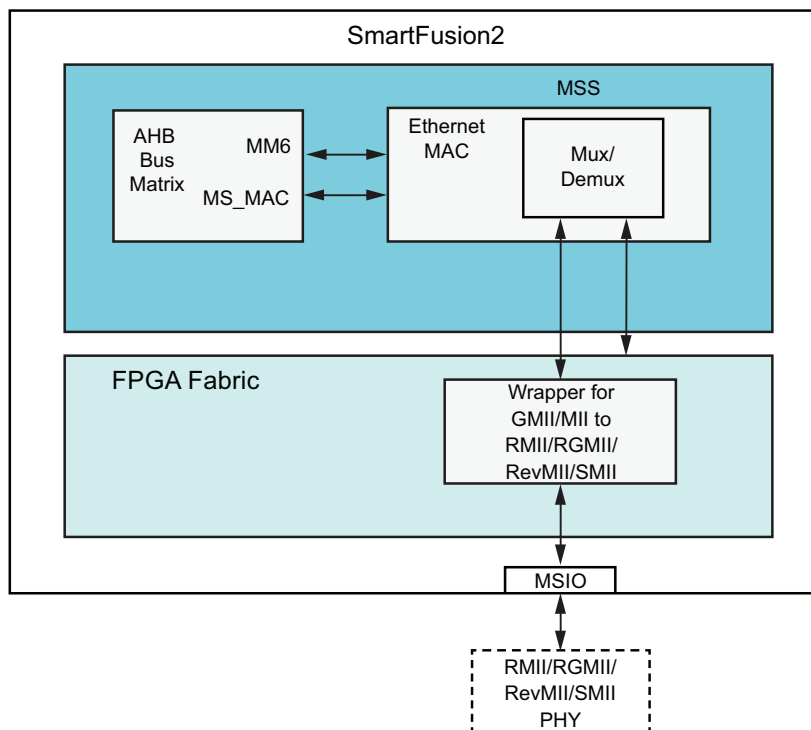
Table 322 • TBI Ports

Port Name	Port Group	Direction	Description
TBI_RCGF[9:0]	MAC_TBI_FABRIC	In	MAC_RCGF is the 10-bit parallel receive data. The receive data byte 0 containing the comma character is byte aligned to 53, 125 MHz receive byte clock used to latch the bytes 0 and 2 of the receive data word.
TBI_TCGF[9:0]	MAC_TBI_FABRIC	Out	MAC_TCGF is the 10-bit parallel transmit data presented in the physical layer for serialization and transmission. The order of transmission is MAC_TCGF[0] first, followed by MAC_TCGF[1] through MAC_TCGF[9].
TBI_RX_CLKP0	MAC_TBI_FABRIC	In	TBI_RX_CLKP0 and TBI_RX_CLKP1 are fed in from the fabric as two 62.5 MHz clocks, which are 180° out of phase with one another.
TBI_RX_CLKP1	MAC_TBI_FABRIC	In	Indicates 125 MHz clocks, which are 180° out of phase with one another.
TBI_GTX_CLK	MAC_TBI_FABRIC	In	Indicates 125 MHz transmit clock from the fabric for 1000 Mbps mode.
TBI_MDI		In	Indicates TBI management data clock
TBI_MDO		Out	Indicates TBI management data output enable
TBI_MDO_EN		Out	Indicates TBI management data out
TBI_MDC		Out	Indicates TBI management data input

Note: Port names have the name of the MAC instance as a prefix, for example: MAC_GMII_TXD.

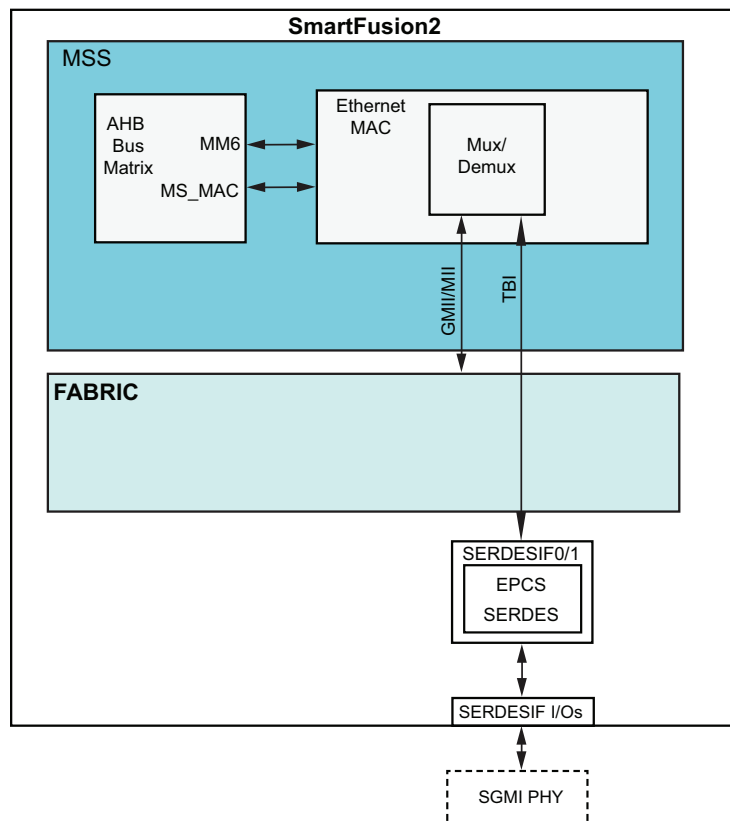
The following figure depicts the RMII, RGMII, RTBI, RevMII, and SMII derived from the available protocols by the appropriate wrapper in the fabric.

Figure 154 • RMII, RGMII, RTBI, RevMII, SMII Derived from Available Protocols by Appropriate Wrapper in Fabric



The following figure depicts TBI to SERDES (EPCS Mode) for SGMII Interface.

Figure 155 • TBI Brought to Fabric for EPCS Soft IP for SGMII Interface



11.4 EMAC Operation

Before any DMA transfers can be carried out, two sets of *descriptors* are needed to be initialized in the host memory. One descriptor is for the transmit operations and the other is for the receive operations. Each set of descriptors takes the form of a linked list typically closed to form a ring buffer.

For ease of handling by software, the transfers are handled using linked lists of transmit and receive descriptors. Transmit and receive descriptors define the buffer in the host memory for Tx operations and another for Rx operations.

The transfer of data in either direction typically uses a ring buffer defined within host memory. The ring buffer for the transmit operations is defined by a closed linked list of the Tx descriptors. The ring buffer for the receive operations is defined by a closed link list of the Rx descriptors. The descriptors act as pointers to the ring buffers. There are separate list of descriptors for both the transmit and receive processes. Each descriptor is in the host memory.

The two ring buffers are formed of an equal-sized segment, each of which is 32-bit aligned and is capable of storing a packet of up to the maximum size of packet transferred. Due to a limitation in the AHB-DMA controller, Ethernet jumbo frames are not supported.

The software can either use the DMA Interrupts generated or poll semaphore bits within the descriptors to maintain synchronization with the packet streams. The entry point into the buffer, used at the start of any sequence of transfers, is given by the descriptor picked out by the DMATx/RxDescriptor register.

Each descriptor comprises a sequence of three 32-bit memory locations as shown in the following table.

Table 323 • Tx/Rx Descriptor

Address	Register	Function	Size
0x0	PacketStartAddr	Start address for the packet data.	32 bits
0x4	Packet Size	Size of packet, Overrides and Empty Flag	32 bits
0x8	Next Descriptor	Location of next descriptor	32 bits

Table 324 • PacketStartAddr

Bit Number	Name	Reset Value	Description
[31:2]	PacketStartAddr[31:2]	0x0	Start address of the packet.
[1:0]	PacketStartAddr[1:0]	0x0	All transfers are 32-bit aligned in host memory.

Table 325 • Packet Size

Bit Number	Name	Reset Value	Description
31	Empty Flag	0x0	For the transmit operations, this bit indicates the availability of the data associated with the packet. For the receive operations, this bit indicates the availability of the specified location to store the received packet. The setting of this flag is used to validate the descriptor.
[30:21]	Reserved	0x0	Reserved

Table 325 • Packet Size (continued)

[20:16]	FTPP Overrides	0x0	The 5-bit field containing the FIFO transmit per-packet override flags signaled to the A-MCXFIFO during the packet transmission. The bits are encoded as follows: 20: FIFO transmit control frame flag. 19:18: FIFO transmit per-packet pad mode flag. 0x0: Do not pad transmit frame. 0x1: Pad all frames to 64 bytes and append FCS to all frames. 0x2: Reserved. 0x3: Reserved. 17: FIFO transmit per-packet generate FCS flag. 16: FIFO transmit per-packet enable flag.
[15:12]	Reserved	0x0	Reserved.
[11:0]	PacketSize	0x0	For the transmit operations, the 12-bit field gives the size of packet to be transferred in bytes. In the receive operations, the DMA controller writes the number of bytes received to this field. The value of this field prior to the transfer being made is ignored.

Table 326 • Next Descriptor

Bit Number	Name	Reset Value	Description
[31:2]	Next Descriptor[31:2]	0x0	The built-in DMA controller reads this register to discover the location in the host memory of the descriptor for the next packet in the sequence. The descriptors should form a closed linked list.
[1:0]	Next Descriptor[1:0]	0x0	All descriptors are 32-bit aligned in the host memory.

11.4.1 Transmit Operation

- Before any packet can be transmitted, a group of Tx descriptors needs to be set up to define the ring buffer used for transmit operations.
The start addresses set for the different segments of the ring buffer are required to be word aligned and should be spaced to give segments of equal size, each able to handle a packet of the maximum size to be transferred.
The packet size component of transmit descriptors should initially be written to have '1' in bit 31, which is the empty flag to indicate that the ring buffer does not currently contain any valid data.
- The least significant four bits of the DMA interrupt mask register are set to specify which Tx DMA events cause a DMA interrupt to be generated.
- The data for one or more transmit packets should then be placed in contiguous segments of the ring buffer. The PacketSize component of the descriptor associated with these segments amended both to record the size of the packet placed in the buffer and to set the Empty Flag to '0' to indicate the presence of valid data.
- The location of the descriptor, which acts as the entry point in the Tx ring buffer, is written in the DMA Tx descriptor register. The DMA transfer of the transmit packets are enabled by writing a '1' to the bit 0 of the DMA Tx control register, which is Tx enable bit.
- The built-in DMA controller then reads the DMA Tx descriptor register to discover the location of the first Tx descriptor. The Tx descriptor is read to check the validity of the associated packet which is indicated by the empty flag, the start address of the packet to be transmitted, and its size.
If the empty flag is '1' then the descriptor is not associated with valid data. The DMA controller terminates the sequence of transmit packet transfers, set the TxUnderrun bit in the DMA Tx Status register and clear the TxEnable bit in the DMA Tx Control register. The TxUnderrun bit in the DMA Tx Status register is set whenever DMA controller reads a '1' in the empty flag of the Tx Descriptor being processed.

If the empty flag is '1', the DMA terminates the transmit operation and then an interrupt is generated for TxUnderrun if enabled. The DMA interrupts register shows TxUnderrun as the source of this interrupt. Any further transfers require the DMA Tx descriptor register to be updated to record the start position in the ring buffer and to set the TxEnable bit to '1' again.

6. The transfer starts when the FIFO indicates that there is a space in the FIFO for a packet of the maximum packet size.
7. If the transfer is completed successfully, the DMA controller writes '1' to bit 31 of the PacketSize component of the descriptor. The TxPktSent flag in the DMA_TX_STATUS register needs to be set (if not already set), the TxPktSent interrupt is to be generated (if enabled) and the TxPktCount is recorded in bits [23:16] of the DMA_TX_STATUS register incremented by 1.
8. The DMA controller then moves on to process any packet stored in the next segment of the ring buffer. The location of the descriptor associated with the next segment in the ring has already been read from the NextDescriptor component of the sequence of transmit packet transfers, sets the Bus Error bit in the DMA_TX_STATUS register, and clears the TxEnable bit in the DMA_TX_CTRL register. If enabled, an interrupt is generated with the DMA Interrupts register showing a Tx Bus Error as the source of this interrupt.

Any further transfers require the DMA_TX_DESC register to be updated to record the new start position in the ring buffer and the TxEnable bit to be set to '1' again.

11.4.2 Receive Operation

1. Before any packets can be received, a group of Rx descriptors needs to be set up to define the ring buffer used for receive operations.
The start addresses set for the different segments of the ring buffer are required to be word aligned and should be spaced to give segments of equal size, each able to handle a packet of the maximum size to be transferred.

The packet size components of these descriptors should initially be written to have '1' in bit 31 which is the empty flag to indicate that the ring buffer does not currently contain any received packets.

2. Bits [7:4] of the DMA Interrupt Mask register are set to specify which Rx DMA events cause a DMA interrupt to be generated.
3. The location of the descriptor corresponding to the entry point in the Rx ring buffer should be written to the DMA Rx descriptor register and to enable the DMA transfer of receive packets enabled by writing a '1' the bit 0 of DMA Rx control register which is the RxEnable bit.
4. The built-in DMA controller then reads DMA Rx descriptor to discover the location of the first Rx descriptor. The built-in DMA controller then reads that descriptor to check that the associated area of the host memory is available for storing the received packet. This is indicated by the empty flag in bit 31 of the PacketSize component of the descriptor and the start address of this storage area.
If the Empty Flag is '0', this suggests that the storage area already contains a packet that has not yet been read by the host software.

When the Empty flag is '0', the DMA controller terminates the sequence of the receive packet transfers, sets the RxOverflow bit in the DMA_RX_STATUS register and clears the RxEnable bit in the DMA_RX_CTRL register.

If enabled, an interrupt is generated with the DMA Interrupts register showing RxOverflow as the source of this interrupt. Any further transfers require the DMA_RX_DESC register to be updated to record the start position in the ring buffer that is now required and the RxEnable bit to be set to '1' again.

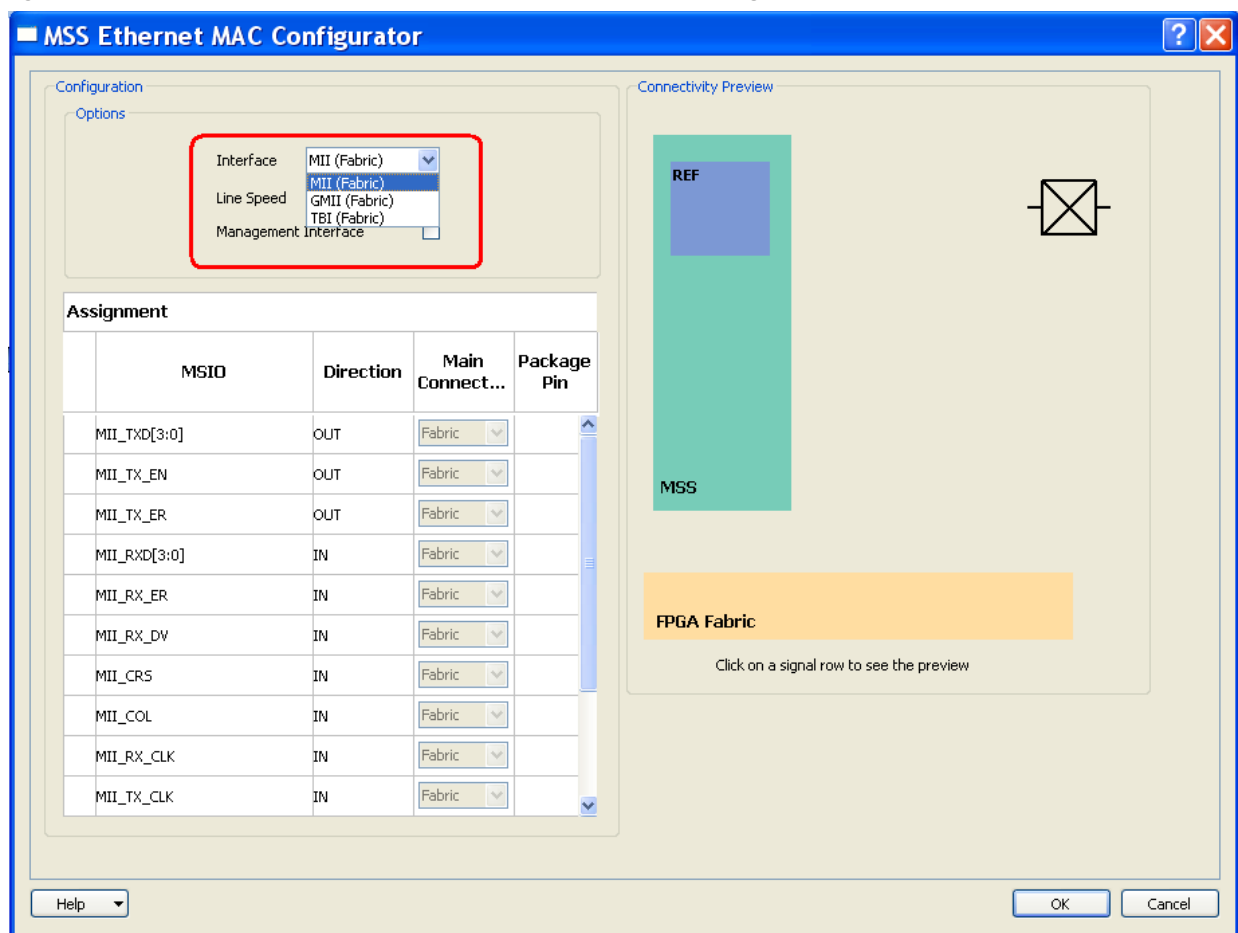
5. Transfer starts when FIFO indicates that there is a packet waiting to be transferred.
6. If the transfer is completed successfully, the DMA controller records the number of bytes transmitted in bits [11:0] of the PacketSize component of the descriptor and writes '0' in bit 31 to record that a packet has been stored in the ring buffer.
The RxPktReceived flag in the DMA_RX_STATUS register is also to be set (if not already set), the RxPktReceived interrupt is to be generated (if enabled) and the RxPktCount is recorded in bits[23:16] of that register incremented by 1.
7. The DMA controller then moves on to transfer the next packet in the next segment of the ring buffer. The location of the descriptor associated with the next segment in the ring has already been read from the NextDescriptor component of the current descriptor.

8. The software should respond to the RxPktReceived interrupt by reading the packet from its location in the ring buffer and then setting the Empty Flag in the descriptor to '1' again to mark this segment of the ring buffer as available for storing further received packets.
9. If a bus error occurs, the DMA controller terminates the sequence of the receive packet transfers, sets the Bus Error bit in the DMA_RX_STATUS register, and clears the RxEnable bit in the DMA_RX_CTRL register. If enabled, an interrupt is generated with the DMA Interrupts register showing an Rx Bus Error as the source of this interrupt.
Any further transfers require the DMA_RX_DESC register to be updated to record the start position in the ring buffer that is now required and the RxEnable bit is to be set to '1' again.

11.5 How to Use TSEMAC

TSEMAC can be configured using the Libero SoC design software. Using the MSS Ethernet Configurator macro, external PHY interface can be selected as shown in the following figure. The external PHY interface can be MII or GMII or TBI. The MII and GMII interfaces are routed through the FPGA fabric onto the MSIOs, and the TBI interface is routed through the FPGA fabric on-to the SERDESIOs.

Figure 156 • External PHY Interface Selection in MSS EMAC Configurator

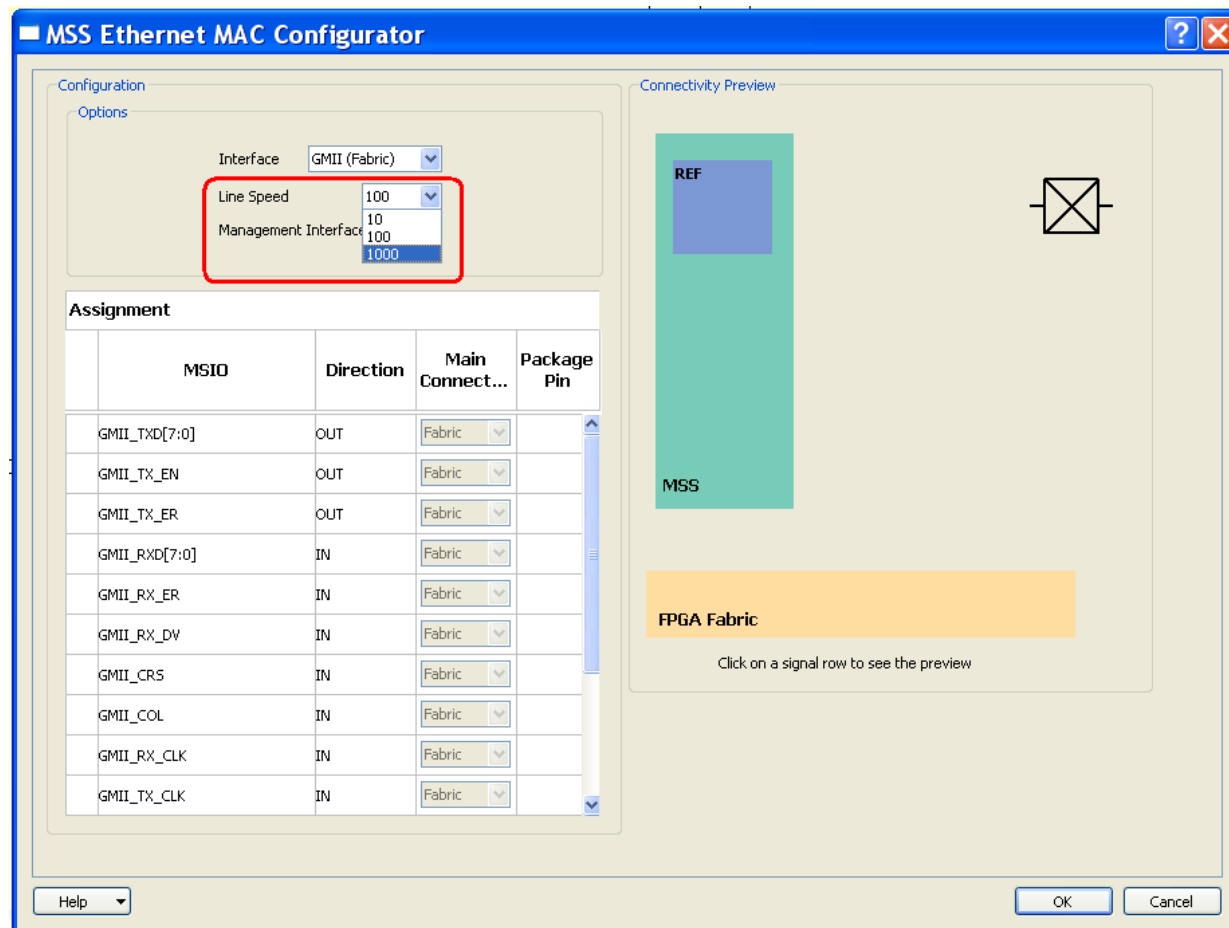


Using the MSS EMAC configurator, the line speed can be selected. Supported line speeds are:

- 10/100 Mbps for the MII interface
- 10/100/1000 Mbps for the GMII and TBI interfaces

The following figure shows how to select the line speed for the selected interface.

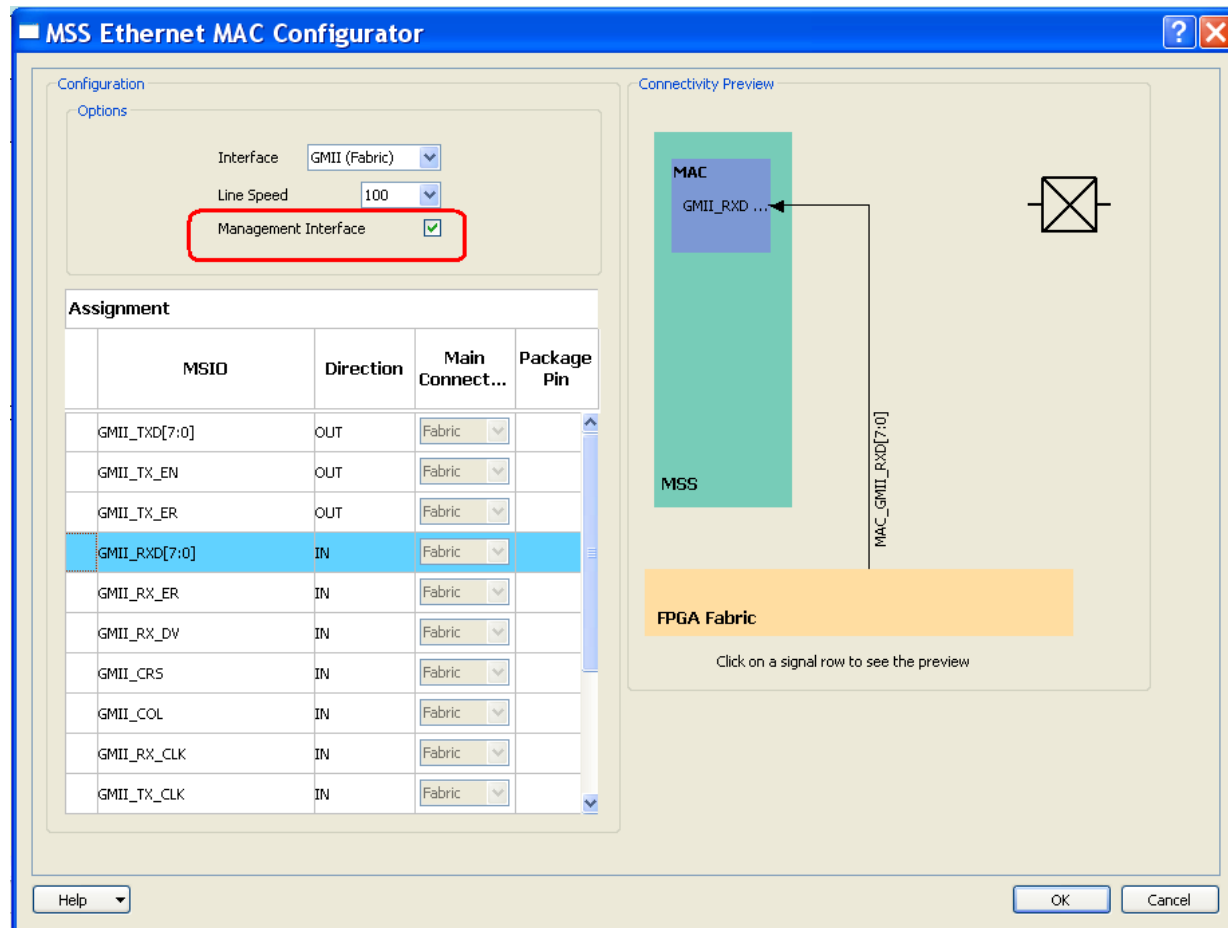
Figure 157 • Line Speed Selection in MSS EMAC Configurator



Using the MSS EMAC configurator, the management interface can be selected. The management interface is used to exchange the control and status information with the external PHY.

The following figure shows how to select the PHY management interface.

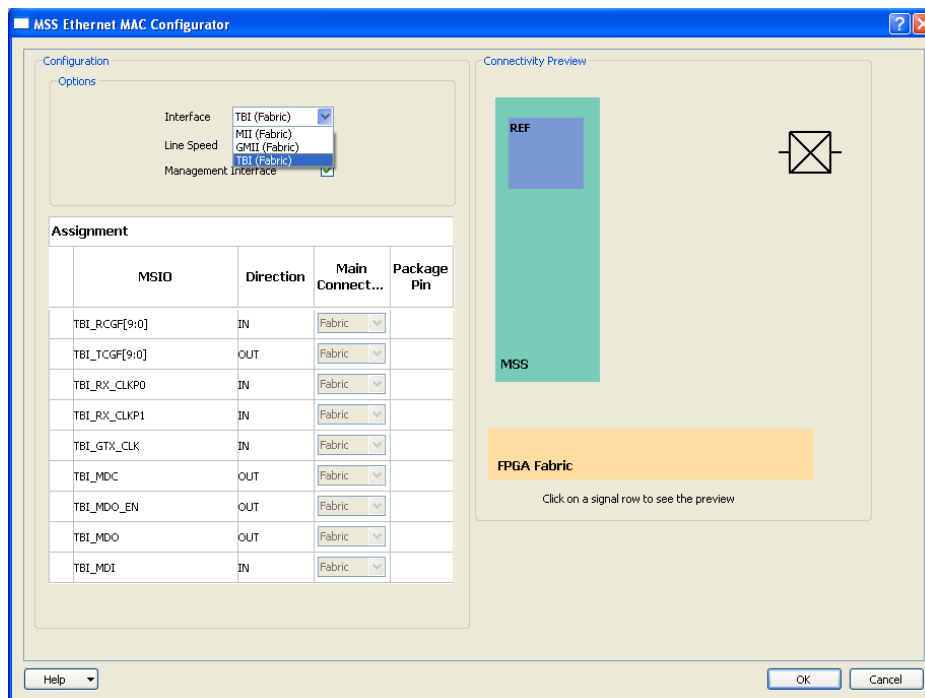
Figure 158 • External PHY Management Interface Selections in MSS EMAC Configurator



11.5.1 SGMII Interface Configuration

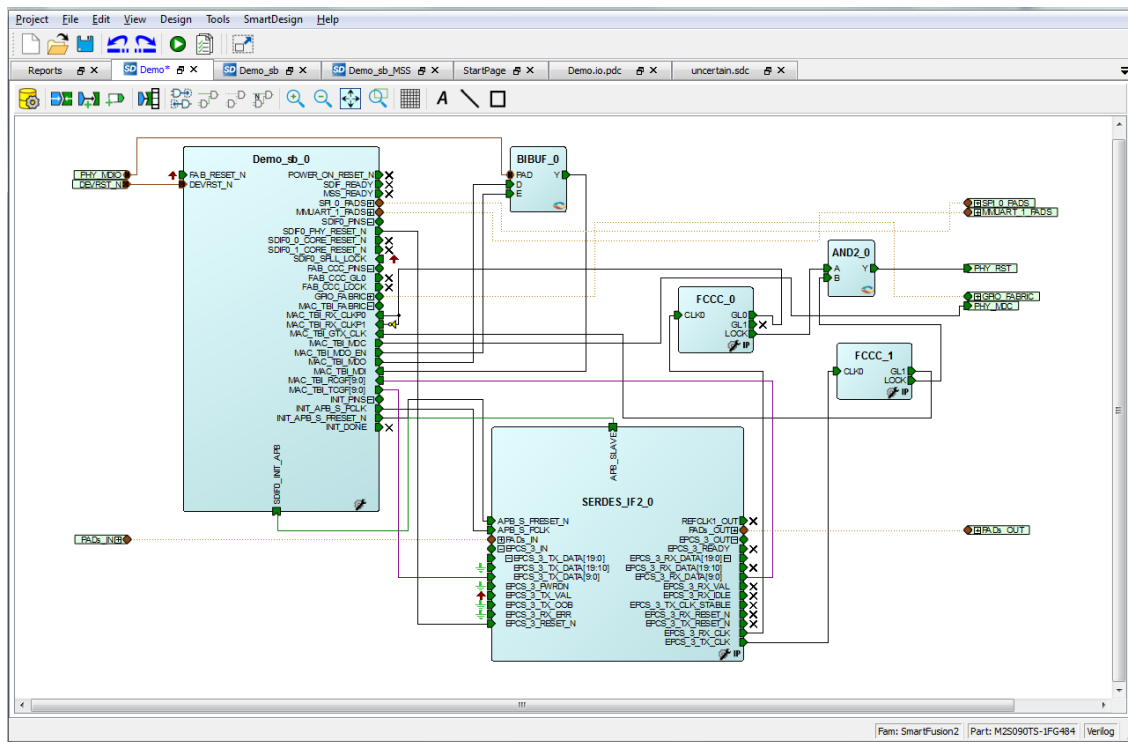
1. Select the interface as TBI, line speed as required. Enable the management interface check box as shown in the following figure.

Figure 159 • MSS Ethernet Configurator with TBI interface



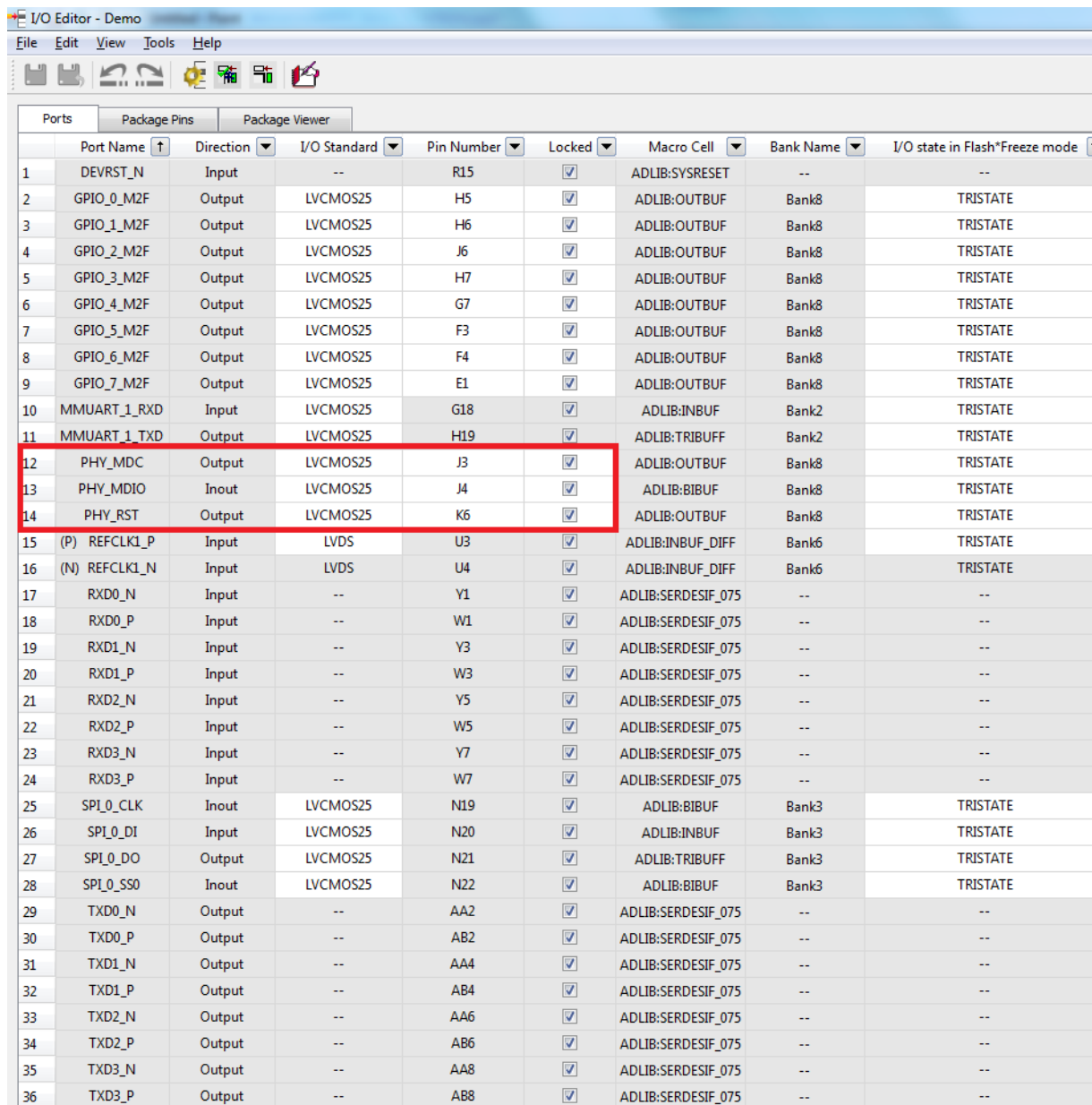
2. Connect TBI signals to SERDES, which is configured for EPCS mode, as shown in the following figure.

Figure 160 • SGMII Interface Signals: TBI to SERDES



3. In design flow window of Libero SoC under compile option open Edit I/O attributes option and assign pin names to PHY interface as shown in the following figure.

Figure 161 • I/O Editor With SGMII and PHY Ports



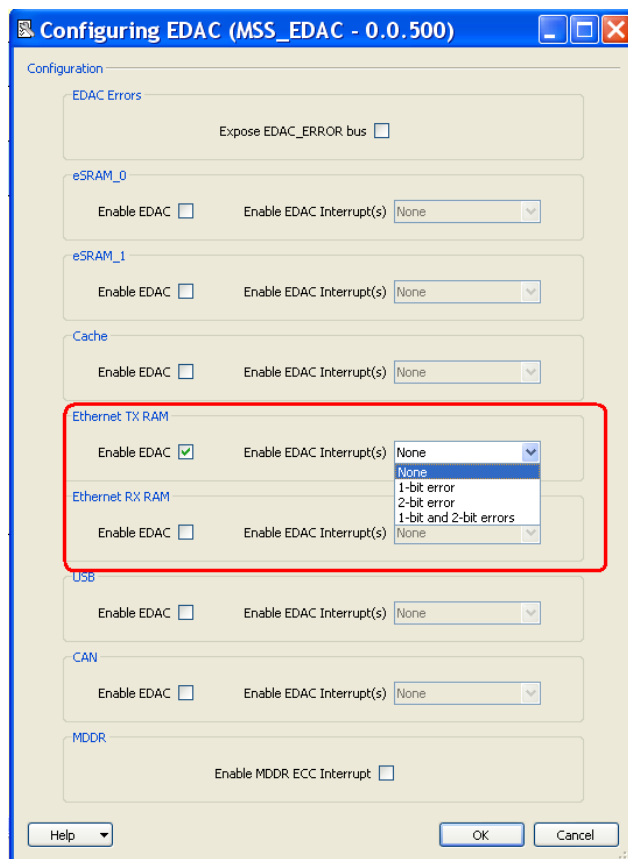
Ports	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze mode
1	DEVRST_N	Input	--	R15	<input checked="" type="checkbox"/>	ADLIB:SYSRESET	--	--
2	GPIO_0_M2F	Output	LVC MOS25	H5	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
3	GPIO_1_M2F	Output	LVC MOS25	H6	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
4	GPIO_2_M2F	Output	LVC MOS25	J6	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
5	GPIO_3_M2F	Output	LVC MOS25	H7	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
6	GPIO_4_M2F	Output	LVC MOS25	G7	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
7	GPIO_5_M2F	Output	LVC MOS25	F3	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
8	GPIO_6_M2F	Output	LVC MOS25	F4	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
9	GPIO_7_M2F	Output	LVC MOS25	E1	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
10	MMUART_1_RXD	Input	LVC MOS25	G18	<input checked="" type="checkbox"/>	ADLIB:INBUF	Bank2	TRISTATE
11	MMUART_1_TXD	Output	LVC MOS25	H19	<input checked="" type="checkbox"/>	ADLIB:TRIBUFF	Bank2	TRISTATE
12	PHY_MDC	Output	LVC MOS25	J3	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
13	PHY_MDIO	Inout	LVC MOS25	J4	<input checked="" type="checkbox"/>	ADLIB:BIBUF	Bank8	TRISTATE
14	PHY_RST	Output	LVC MOS25	K6	<input checked="" type="checkbox"/>	ADLIB:OUTBUF	Bank8	TRISTATE
15	(P) REFCLK1_P	Input	LVDS	U3	<input checked="" type="checkbox"/>	ADLIB:INBUF_DIFF	Bank6	TRISTATE
16	(N) REFCLK1_N	Input	LVDS	U4	<input checked="" type="checkbox"/>	ADLIB:INBUF_DIFF	Bank6	TRISTATE
17	RXD0_N	Input	--	Y1	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
18	RXD0_P	Input	--	W1	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
19	RXD1_N	Input	--	Y3	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
20	RXD1_P	Input	--	W3	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
21	RXD2_N	Input	--	Y5	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
22	RXD2_P	Input	--	W5	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
23	RXD3_N	Input	--	Y7	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
24	RXD3_P	Input	--	W7	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
25	SPI0_CLK	Inout	LVC MOS25	N19	<input checked="" type="checkbox"/>	ADLIB:BIBUF	Bank3	TRISTATE
26	SPI0_DI	Input	LVC MOS25	N20	<input checked="" type="checkbox"/>	ADLIB:INBUF	Bank3	TRISTATE
27	SPI0_DO	Output	LVC MOS25	N21	<input checked="" type="checkbox"/>	ADLIB:TRIBUFF	Bank3	TRISTATE
28	SPI0_SS0	Inout	LVC MOS25	N22	<input checked="" type="checkbox"/>	ADLIB:BIBUF	Bank3	TRISTATE
29	TXD0_N	Output	--	AA2	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
30	TXD0_P	Output	--	AB2	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
31	TXD1_N	Output	--	AA4	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
32	TXD1_P	Output	--	AB4	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
33	TXD2_N	Output	--	AA6	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
34	TXD2_P	Output	--	AB6	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
35	TXD3_N	Output	--	AA8	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--
36	TXD3_P	Output	--	AB8	<input checked="" type="checkbox"/>	ADLIB:SERDESIF_075	--	--

11.5.2 SECDED Features for TSEMAC Buffers

The 4 KB TX buffer and 8 KB RX buffer are protected with the SECDED feature. Using the EDAC configurator, the SECDED feature can be enabled independently for TX and RX buffers. As shown in the following figure, the SECDED feature for the EMAC TX and RX buffers can be enabled or disabled. Interrupts for single bit error, dual bit error, or both single bit and dual bit errors can be enabled.

To clear these interrupts and to take necessary actions in case of dual bit error, register the interrupt handler in the application code.

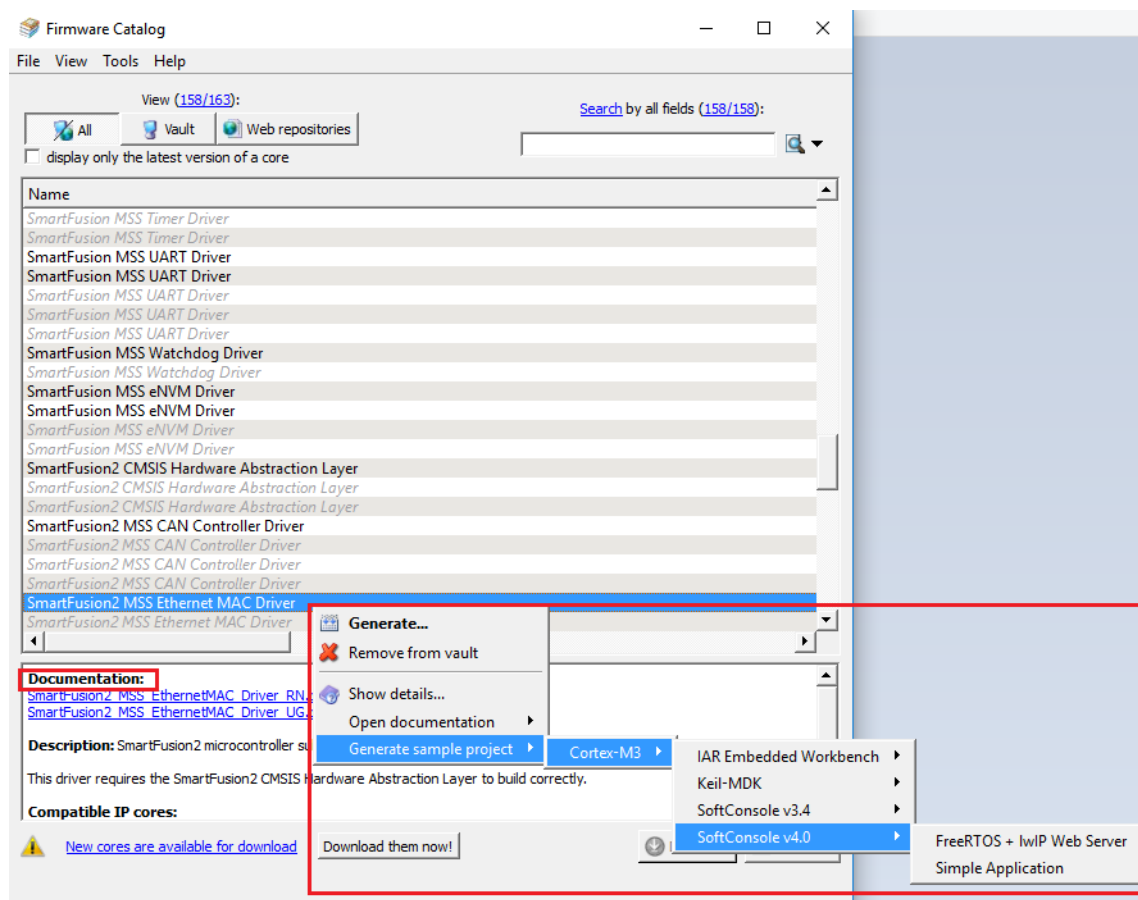
Figure 162 • SECDED Configurator with Ethernet TX RAM and Ethernet RX RAM Configuration Options



Microsemi provides TSEMAC firmware drivers to use with the application development. The SmartFusion2 TSEMAC firmware drivers can be downloaded from the Firmware Catalog. The TSEMAC firmware drivers provides APIs-to-TSEMAC services. Refer to the **SmartFusion2 TSEMAC Driver User Guide** for the list of APIs and their descriptions.

The TSEMAC driver package includes sample projects to show the usage of TSEMAC. The sample projects are available for three different tool chains: IAR Embedded Work, Keil MDK, and SoftConsole. A sample project can be generated by right-clicking the system services driver and selecting **Generate sample project**, as shown in the following figure.

Figure 163 • Firmware Catalog Showing the Generation of Sample Project for TSEMAC



Microsemi provides the device drivers for the TSEMAC controller for SmartFusion2 device and recommends using these drivers for the application development.

The following table lists the APIs available in TSEMAC firmware drivers for initialization and configuration.

Table 327 • TSEMAC Firmware Drivers for Initialization and Configuration

API	Description
MSS_MAC_cfg_struct_def_init	The MSS_MAC_cfg_struct_def_init() function initializes a mss_mac_cfg_t configuration data structure to default values. The default configuration uses the MII interface connected to a PHY at address 0x00 which is set to auto-negotiate at all available speeds up to 1000Mbps. This default configuration can then be used as parameter to MSS_MAC_init(). Typically the default configuration would be modified to suit the application before being passed to MSS_MAC_init()
MSS_MAC_init	The MSS_MAC_init() function initializes the Ethernet MAC hardware and driver internal data structures. The MSS_MAC_init() function takes a pointer to a configuration data structure of type mss_mac_cfg_t as parameter. This configuration data structure contains all the information required to configure the Ethernet MAC. The MSS_MAC_init() function initializes the descriptor rings and their pointers to initial values. The MSS_MAC_init() function enables DMA Rx packet received and Tx packet sent interrupts. The configuration passed to the MSS_MAC_init() function specifies the type of interface used to connect the Ethernet MAC and Ethernet PHY as well as the PHY MII management interface address. It also specifies the allowed link speed and duplex mode. It is at this point that the application chooses if the link speed and duplex mode will be auto-negotiate with the link partner or forced to a specific speed and duplex mode.

The following table lists the APIs available in TSEMAC firmware drivers for Ethernet PHY management.

Table 328 • TSEMAC Firmware Drivers for Ethernet PHY Management

API	Description
MSS_MAC_read_phy_reg	The MSS_MAC_read_phy_reg() function reads the Ethernet PHY register specified as parameter. It uses the MII management interface to communicate with the Ethernet PHY. This function is used part of the Ethernet PHY drivers provided alongside the Ethernet MAC driver. You only need to use this function if writing your own Ethernet PHY driver.
MSS_MAC_write_phy_reg	The MSS_MAC_write_phy_reg() function writes a 16-bit value to the specified Ethernet PHY register. . It uses the MII management interface to communicate with the Ethernet PHY. This function is used part of the Ethernet PHY drivers provided alongside the Ethernet MAC driver. You only need to use this function if writing your own Ethernet PHY driver.

The following table lists the APIs available in TSEMAC firmware drivers for transmit and receive operations

Table 329 • TSEMAC Firmware Drivers for Transmit and Receive Operations

API	Description
MSS_MAC_send_pkt	The MSS_MAC_send_pkt() function initiates the transmission of a packet. It places the buffer containing the packet to send into one of the Ethernet MAC's transmit descriptors. This function is non-blocking. It will return immediately without waiting for the packet to be sent. The Ethernet MAC driver indicates that the packet is sent by calling the transmit completion handler registered by a call to MSS_MAC_set_tx_callback().

Table 329 • TSEMAC Firmware Drivers for Transmit and Receive Operations (continued)

API	Description
MSS_MAC_receive_pkt	<p>The MSS_MAC_receive_pkt() function assigns a buffer to one of the Ethernet MAC's receive descriptors. The receive buffer specified as parameter will be used to receive one single packet. The receive buffer will be handed back to the application via a call to the receive callback function assigned through a call to MSS_MAC_set_rx_callback(). The MSS_MAC_receive_pkt() function will need to be called again pointing to the same buffer if more packets are to be received into this same buffer after the packet has been processed by the application. The MSS_MAC_receive_pkt() function is non-blocking. It will return immediately and does not wait for a packet to be received. The application needs to implement a receive callback function to be notified that a packet has been received.</p> <p>The p_user_data parameter can be optionally used to point to a memory management data structure managed by the application.</p>
MSS_MAC_set_tx_callback	The MSS_MAC_set_tx_callback() function registers the function that will be called by the Ethernet MAC driver when a packet has been sent.
MSS_MAC_set_rx_callback	The MSS_MAC_set_rx_callback() function registers the function that will be called by the Ethernet MAC driver when a packet is received.

The following table lists the APIs available in TSEMAC firmware drivers for reading status and statistics.

Table 330 • TSEMAC Firmware Drivers for Reading Status and Statistics

API	Description
MSS_MAC_read_stat	The MSS_MAC_read_stat() function reads the transmit and receive statistics of the Ethernet MAC. This function can be used to read one of 17 receiver statistics, 20 transmitter statistics and 7 frame type statistics as defined in the mss_mac_stat_t enumeration.
MSS_MAC_clear_statistics	The MSS_MAC_clear_statistics() function clears all the statistics counter registers.
MSS_MAC_get_link_status	<p>The MSS_MAC_get_link_status () function retrieves the status of the link from the Ethernet PHY. It returns the current state of the Ethernet link. The speed and duplex mode of the link is also returned via the two pointers passed as parameter if the link is up.</p> <p>This function also adjusts the Ethernet MAC's internal configuration if some of the link characteristics have changed since the previous call to this function.</p>

Refer to **G4Main MSS Ethernet MAC driver User's Guide** for more information on the API detailed description and parameters to the APIs.

Note: The MSS Ethernet does not support full behavioral simulation models. Refer to *SmartFusion2 MSS BFM Simulation User Guide* for more information.

11.6 SYSREG Control Register for EMAC

The registers are located in the SYSREG section and are listed again here for clarity. Refer to the [System Register Block](#), page 670 for a detailed description of each register and bit.

[Table 320](#), page 378 shows the control information from the MAC_CR register in the system register.

The following table depicts the MAC_CR Register in the SYSREG block.

Table 331 • MAC_CR Register in SYSREG Block

Register Name	Register Type	Flash Write Protect	Reset Source	Description
MAC_CR	RW-P	Register	sysreset_n	MAC configuration register

11.7 EMAC Configuration Register Summary

Each descriptor comprises a sequence of three 32-bit memory locations as shown in [Table 323](#), page 381.

The following table summarizes each of the registers covered in this document. The EMAC base address is 0x40041000.

Table 332 • EMAC M-AHB Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
DMA_TX_CTRL	0x180	R/W	0x0	Transmit control register
DMA_TX_DESC	0x184	R/W	0x0	Pointer to transmit descriptor
DMA_TX_STATUS	0x188	R/W	0x0	Transmit status register
DMA_RX_CTRL	0x18C	R/W	0x0	Receive control register
DMA_RX_DESC	0x190	R/W	0x0	Pointer to receive descriptor
DMA_RX_STATUS	0x194	R/W	0x0	Receive status register
DMA_IRQ_MASK	0x198	R/W	0x0	Interrupt mask register
DMA_IRQ	0x19C	RO	0x0	Interrupts register

Table 333 • EMAC PE-MCXMAC Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
CFG1	0x00	R/W	0x80000000	MAC configuration register
CFG2	0x04	R/W	0x00007000	MAC configuration register
IFG	0x08	R/W	0x40605060	Inter packet gap and interframe gap register
HALF_DUPLEX	0x0C	R/W	0x00A1F037	Definition of half duplex register
MAX_FRAME_LENGTH	0x10	R/W	0x00000600	Sets the maximum frame size in both transmit and receive directions.
Reserved	0x14	R/W	0x0	Reserved
Reserved	0x18	R/W	0x0	Reserved
TEST	0x1C	R/W	0x0	This test bit is used to predict back off times in Half-duplex mode. This allows the MAC to be paused for testing purpose only.
MII_CONFIG	0x20	R/W	0x0	This resets MII MGMT, determines MGMT Clock frequency and causes the MII MGMT to suppress preamble generation.
MII_COMMAND	0x24	R/W	0x0	MONITORS link fails.
MII_ADDRESS	0x28	R/W	0x0	This represents 5-bit PHY address field and 5 bit register address field.
MII_CTRL	0x2C	WO	0x0	Control register for MII management write cycle using the 16-bit data and the pre-configured PHY and register addresses.

Table 333 • EMAC PE-MCXMAC Register Map (continued)

Register Name	Address Offset	Register Type	Reset Value	Description
MII_STATUS	0x30	RO	0x0	Following an MII Mgmt read cycle, the 16-bit data can be read from this location.
MII_INDICATORS	0x34	RO	0x0	This indicates MII management block is currently performing an MII Mgmt read or write cycle.
INTERFACE_CTRL	0x38	R/W	0x0	This configures PERMII for 10 Mbps or 100 Mbps speed.
INTERFACE_STATUS	0x3C	RO	0x0	This indicates the serial MII PHY has detected a jabber condition on the link. This also indicates the serial MII PHY has detected a valid link. This indicates the serial MII PHY is operating in Full-duplex mode.
STATION_ADDRESS1	0x40	R/W	0x0	The register fields hold the station address. Station address is the 48-bit programmed receive frame's destination address.
STATION_ADDRESS2	0x44	R/W	0x0	The register fields hold the station address. The station address is the 48-bit programmed receive frame's destination address.

Table 334 • EMAC A-MCXFIFO Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
FIFO_CFG0	0x48	R/W	0x0	Definition of A-MCXFIFO configuration register 0
FIFO_CFG1	0x4C	R/W	0x0FFFFFFF	Definition of A-MCXFIFO configuration register 1
FIFO_CFG2	0x50	R/W	0x1FFF1FFF	Definition of A-MCXFIFO configuration register 2
FIFO_CFG3	0x54	R/W	0xFFF0FFF	Definition of A-MCXFIFO configuration register 3
FIFO_CFG4	0x58	R/W	0x0	Definition of A-MCXFIFO configuration register 4
FIFO_CFG5	0x5C	R/W	0x3FFFF	Definition of A-MCXFIFO configuration register 5
FIFO_RAM_ACCESS0	0x60	R/W	0x0	The FIFO RAM access register 0 is intended for non-real-time RAM testing and debug.
FIFO_RAM_ACCESS1	0x64	R/W	0x0	The FIFO RAM access register 1 is intended for non-real-time RAM testing and debug.
FIFO_RAM_ACCESS2	0x68	R/W	0x0	The FIFO RAM access register 2 is intended for non-real-time RAM testing and debug.
FIFO_RAM_ACCESS3	0x6C	RO	0x0	The FIFO RAM access register 3 is intended for non-real-time RAM testing and debug.
FIFO_RAM_ACCESS4	0x70	R/W	0x0	The FIFO RAM access register 4 is intended for non-real-time RAM testing and debug.
FIFO_RAM_ACCESS5	0x74	R/W	0x0	The FIFO RAM access register 5 is intended for non-real-time RAM testing and debug.

Table 334 • EMAC A-MCXFIFO Register Map (continued)

Register Name	Address Offset	Register Type	Reset Value	Description
FIFO_RAM_ACCESS6	0x78	R/W	0x0	The FIFO RAM access register 6 is intended for non-real-time RAM testing and debug.
FIFO_RAM_ACCESS7	0x7C	RO	0x0	The FIFO RAM access register 7 is intended for non-real-time RAM testing and debug.

Table 335 • EMAC PE-MSTAT Transmit and Receive Counters Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
TR64	0x80	R/W	0x0	Transmit and receive 64 byte frame counter
TR127	0x84	R/W	0x0	Transmit and receive 65 to 127 byte frame counter
TR255	0x88	R/W	0x0	Transmit and receive 128 to 255 byte frame counter
TR511	0x8C	R/W	0x0	Transmit and receive 256 to 511 byte frame counter
TR1K	0x90	R/W	0x0	Transmit and receive 512 to 1023 byte frame counter
TRMAX	0x94	R/W	0x0	Transmit and receive 1024 to 1518 byte frame counter
TRMGV	0x98	R/W	0x0	Transmit and receive 1519 to 1522 byte good VLAN frame count

Table 336 • EMAC PE-MSTAT Receive Counters Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
RBYT	0x9C	R/W	0x0	The statistic counter register is incremented by the byte count of all frames received.
RPKT	0xA0	R/W	0x0	Incremented for each frame received packet.
RFCS	0xA4	R/W	0x0	This is incremented for each frame received that has an integral 64 to 1518 length and contains a frame check sequence error.
RMCA	0xA8	R/W	0x0	This is incremented for each multicast good frame of lengths smaller than 1518 (non VLAN) or 1522 (VLAN) excluding broadcast frames.
RBCA	0xAC	R/W	0x0	This is incremented for each broadcast good frame of lengths smaller than 1518 (non VLAN) or 1522 (VLAN) excluding multicast frames.
RXCF	0xB0	R/W	0x0	This is incremented for each MAC control frame received.
RXPF	0xB4	R/W	0x0	This is incremented each time a valid PAUSE MAC control frame is received.
RXUO	0xB8	R/W	0x0	This is incremented each time a MAC control frame is received which contains an op code other than a PAUSE.

Table 336 • EMAC PE-MSTAT Receive Counters Register Map (continued)

Register Name	Address Offset	Register Type	Reset Value	Description
RALN	0xBC	R/W	0x0	This is incremented for each received frame from 64 to 1518, which contains an invalid FCS and is not an integral number of bytes.
RFLR	0xC0	R/W	0x0	This is incremented for each frame received in which the 802.3 length field does not match the number of data bytes actually received (46 – 1500 bytes).
RCDE	0xC4	R/W	0x0	This is incremented each time a valid carrier is present and at least one invalid data symbol is detected.
RCSE	0xC8	R/W	0x0	This is incremented each time a false carrier is detected.
RUND	0xCC	R/W	0x0	This is incremented each time a frame is received, which is less than 64 bytes in length and contains a valid frame check sequence (FCS).
ROVR	0xD0	R/W	0x0	This is incremented each time a frame is received which exceeds 1518 (non VLAN) or 1522 (VLAN) bytes and contains a valid FCS.
RFRG	0xD4	R/W	0x0	This is incremented for each frame received which is less than 64 bytes in length and contains an invalid FCS.
RJBR	0xD8	R/W	0x0	This is incremented for frames received which exceed 1518 (non VLAN) or 1522 (VLAN) bytes and contains an invalid FCS.
RDRP	0xDC	R/W	0x0	This is incremented for frames received which are streamed to system but are later dropped due to lack of system resources.

Table 337 • EMAC PE-MSTAT Transmit Counters Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
TBYT	0xE0	R/W	0x0	This is incremented for each transmitted byte including fragments of frames which are involved in collisions.
TPKT	0xE4	R/W	0x0	This is incremented for each transmitted packet.
TMCA	0xE8	R/W	0x0	This is incremented for each transmitted multicast valid frame.
TBCA	0xEC	R/W	0x0	This is incremented for each transmitted broadcast frame.
TXPF	0xF0	R/W	0x0	This is incremented each time a valid PAUSE MAC control frame is transmitted.
TDFR	0xF4	R/W	0x0	This incremented for each frame, which is deferred on its first transmission attempt.
TEDF	0xF8	R/W	0x0	This is incremented for aborted frames, which are deferred for an excessive period of time (3036 byte times).
TSCL	0xFC	R/W	0x0	This is incremented for each transmitted frame that experiences exactly one collision during the transmission.
TMCL	0x100	R/W	0x0	This is incremented for each transmitted frame that experiences 2 to 15 collisions (including any late collisions) during the transmission.
TLCL	0x104	R/W	0x0	This is incremented for each transmitted frame which experiences a late collision during a transmission attempt.

Table 337 • EMAC PE-MSTAT Transmit Counters Register Map (continued)

Register Name	Address Offset	Register Type	Reset Value	Description
TXCL	0x108	R/W	0x0	This is incremented for each frame that experiences 16 collisions during the transmission and is aborted.
TNCL	0x10C	R/W	0x0	This is incremented by the number of collisions experienced during the transmission of a frame.
TPFH	0x110	R/W	0x0	This is incremented each time a valid PAUSE MAC control frame is transmitted and honored.
TDRP	0x114	R/W	0x0	This is incremented each time PAUSE frame is honored.
TJBR	0x118	R/W	0x0	This is incremented for each oversized transmitted frame with an incorrect FCS value.
TFCS	0x11C	R/W	0x0	This is incremented for each valid sized packet with an incorrect FCS value.
TXCF	0x120	R/W	0x0	This is incremented for each valid size frame with a Type Field signifying a Control frame.
TOVR	0x124	R/W	0x0	This is incremented for each oversized transmitted frame with a correct FCS value.
TUND	0x128	R/W	0x0	This is incremented for each frame which is less than 64 bytes with a correct FCS value.
TFRG	0x12C	R/W	0x0	This is incremented for each frame which is less than 64 bytes, with an incorrect FCS value.
CAR1	0x130	RO	0x0	This indicates the transmit and receive counters and the receive counters carry the bits. The carry register bits are cleared on carry register write when the respective bit is asserted.
CAR2	0x134	RO	0x0	This indicates the transmit counters carry bits. The carry register bits are cleared on carry register write when the respective bit is asserted.
CAM1	0x138	R/W	0xFE01FFFF	The rollover condition of each transmit and receive counter and receive counters can be discreetly masked from causing an interrupt by internal masking.
CAM2	0x13C	R/W	0xFFFFF	The rollover condition of each transmit counter can be discreetly masked from causing an interrupt by internal masking.

Table 338 • EMAC M-SGMII Register Map

Register name	Address Offset	Register Type	Reset Value	Description
SGMII CONTROL	0x00	R/W	0x0	This enables the loopback and the auto negotiation. The PHY address for the M-SGMII is 0x1E.
SGMII STATUS	0x01	RO	0x0001	This enables the MF PREMABLE suppression enable. This indicates the auto negotiation complete, associated PHY auto negotiation ability and link status. The PHY address for the M-SGMII is 0x1E.
RESERVED	0x02	R/W	0x0	Reserved

Table 338 • EMAC M-SGMII Register Map (continued)

Register name	Address Offset	Register Type	Reset Value	Description
RESERVED	0x03	R/W	0x0	Reserved
AN SGMII ADVERTISEMENT	0x04	R/W	0x0	This indicates that the link is up when the M-SGMII is integrated into a PHY and is communicating with the SGMII module in a MAC. It also indicates the link is transferring data in Full-duplex mode and link speed. The PHY address for the M-SGMII is 0x1E.
AN LINK PARTNER BASE PAGE ABILITY	0x05	RO	-	This indicates that the link is transferring data in Full-duplex mode. This also indicates the speed of the link. The PHY address for the M-SGMII is 0x1E.
AN EXPANSION	0x06	RO	0x0	This indicates that the device supports the next page function. This also indicates that the new page is received and stored in the applicable AN LINK PARTNER ABILITY or AN NEXT PAGE register. The PHY address for the M-SGMII is 0x1E.
AN NEXT PAGE TRANSMIT	0x07	R/W	0x0	This indicates the additional next pages to follow and message page. Message pages are formatted pages, which carry a predefined message code that is enumerated in IEEE 802.3u/Annex 28C. The PHY address for the M-SGMII is 0x1E.
AN NEXT PAGE TRANSMIT	0x08	RO	-	The link partner asserts this bit to indicate additional Next Pages to follow. This indicates the message page and the link partner's ability to comply with the message. The PHY address for the M-SGMII is 0x1E.
EXTENDED STATUS	0x0F	RO	0xA000	This indicates that the PHY can be operated in 1000BASE-X FULL-DUPLEX, 1000BASE-X HALF-DUPLEX, 1000BASE-T FULL-DUPLEX, 1000BASE-T HALF-DUPLEX. The PHY address for the M-SGMII is 0x1E.
JITTER DIAGNOSTICS	0x10	R/W	0x0	This enables the M-SGMII to transmit the jitter test patterns, which are defined in the IEEE 802.3z 36A. This selects the jitter pattern that is to be transmitted in diagnostics mode. The PHY address for the M-SGMII is 0x1E.
TBI CONTROL	0x11	R/W	0x0	This allows the auto-negotiation function to sense either a gigabit MAC in the auto-negotiation bypass mode or an older gigabit MAC without the auto-negotiation capability. This defines the M-SGMII as being in 1000BASE-X or SERDES mode. This allows the SERDES PHY to perform the code group alignment based upon the detection of a comma. The PHY address for the M-SGMII is 0x1E.

Minimal configuration, required for MAC to make it functional in 1000 Mbps mode of operation, is given below:

```
CFG1= 32'h0000_0035      //Rx/Tx flow control enable, Rx/Tx-Enable
CFG2 = 32'h0000_7202      //Preamble=7, byteMode, CRC-enable
STATION_ADDRESS1 = 32'hA5A4_A3A2      //Station Address 1-4
STATION_ADDRESS2 = 32'hA1A0_0000      //Station Address 5-6
FIFO_CFG0 = 32'h0000_FF00      //Enable FIFO transmit and receive modules
FIFO_CFG3 = 32'h007F_FFFF      // Tx-FIFO high watermark=128
```

11.8 EMAC Register Bit Definitions

The following tables are the bit definitions of the registers present in EMAC.

Table 339 • DMA_TX_CTRL

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0x0	Reserved
0	Transmit control	0x0	TxEnable: Setting this bit enables DMA transmit packet transfers. The bit is cleared by the built-in DMA controller whenever it encounters a Tx Underrun or Bus Error state.

Table 340 • DMA_TX_DESC

Bit Number	Name	Reset Value	Description
[31:2]	Top 30 bits of Descriptor Address	0x0.	When TxEnable is set by the host, the built-in DMA controller reads this register to discover the location in the host memory of the first transmit packet descriptor.
[1:0]	Ignored by the DMA controller	0x0	All descriptors are 32-bit aligned in the host memory.

Table 341 • DMA_TX_STATUS

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0x0	Reserved
[23:16]	TxPktCount	0x0	The 8-bit transmit packet counter that is incremented whenever the built-in DMA controller successfully transfers a packet, and is decremented whenever the host writes a '1' to bit '0' in this register.
[16:4]	Reserved	0x0	Reserved
3	BusError	0x0	When set, this indicates that a host slave split, retry or error response is received by the DMA controller.
2	Reserved	0x0	Reserved
1	TxUnderrun	0x0	Set whenever the DMA controller reads a '1' for the empty flag in the descriptor.
0	TxPktSent	0x0	When set, this indicates that one or more packets have been successfully transferred. Writing a '1' to this bit reduces the TxPktCount value by one. The bit is cleared whenever TxPktCount is zero.

Table 342 • DMA_RX_CTRL

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0x0	Reserved
0	Rx Enable	0x0	Setting this bit enables DMA receive packet transfers. The bit is cleared by the built-in DMA controller whenever it encounters an Rx overflow or bus error state.

Table 343 • DMA_RX_DESC

Bit Number	Name	Reset Value	Description
[31:1]	Top 30 bits of Descriptor Address	0x0	When RxEnable is set by the host, the built-in DMA controller reads this register to discover the location in the host memory of the first receive packet descriptor.
0	Ignored by the DMA controller	0x0	All descriptors are 32-bit aligned in the host memory.

Table 344 • DMA_RX_STATUS

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0x0	Reserved
[23:16]	RxPktCount	0x0	The 8-bit receive packet counter that is incremented whenever the built-in DMA controller successfully transfers a packet, and is decremented whenever the host writes a '1' to bit zero of this register.
[16:4]	Reserved	0x0	Reserved
3	BusError	0x0	When set, this indicates that a host slave split, retry or error response is received by the DMA controller.
2	RxOverflow	0x0	Set whenever the DMA controller reads a zero empty flag in the descriptor it is processing.
1	Reserved	0x0	Reserved
0	RxPktReceived	0x0	When set, this indicates that one or more packets have been successfully transferred. Writing a '1' to this bit reduces the RxPktCount value by one. The bit is cleared whenever RxPktCount is zero.

Table 345 • DMA_IRQ_MASK

Bit Number	Name	Reset Value	Description
[31:8]	Reserved	0x0	Reserved
7	Bus Error Mask	0x0	Setting this bit to '1' enables the Bus Error bit in the DMARxStatus register as an interrupt source.
6	Rx Overflow Mask	0x0	Setting this bit to '1' enables the RxOverflow bit in the DMARxStatus register as an interrupt source.
5	Reserved	0x0	Reserved

Table 345 • DMA_IRQ_MASK (continued)

Bit Number	Name	Reset Value	Description
4	RxPktReceived Mask	0x0	Setting this bit to '1' enables the RxPktReceived bit in the DMARxStatus register as an interrupt source.
3	Bus Error Mask	0x0	Setting this bit to '1' enables the Bus Error bit in the DMATxStatus register as an interrupt source.
2	Reserved	0x0	Reserved
1	Tx Underrun Mask	0x0	Setting this bit to '1' enables the TxUnderrun bit in the DMATxStatus register as an interrupt source.
0	TxPktSent Mask	0x0	Setting this bit to '1' enables the TxPktSent bit in the DMATxStatus register as an interrupt source.

Table 346 • DMA_IRQ

Bit Number	Name	Reset Value	Description
[31:8]	Reserved	0x0	Reserved
7	Bus Error	0x0	This is set to '1' to record a receive bus error interrupt when the Bus Error bit in the DMARxStatus register and bit 7 of the DMAIntrMask register are both set.
6	Rx Overflow	0x0	This is set to '1' to record an Rx overflow interrupt when the RxOverflow bit in the DMARxStatus register and bit 6 of the DMAIntrMask register are both set.
5	Reserved	0x0	Reserved
4	RxPktReceived	0x0	This is set to '1' to record a RxPktReceived interrupt when the RxPktReceived bit in the DMARxStatus register and bit 4 of the DMAIntrMask register are both set.
3	Bus Error	0x0	This is set to '1' to record a transmit bus error interrupt when the Bus Error bit in the DMATxStatus register and bit 3 of the DMAIntrMask register are both set.
2	Reserved	0x0	Reserved
1	Tx Underrun	0x0	This is set to '1' to record a Tx underrun interrupt when the TxUnderrun bit in the DMATxStatus register and bit 1 of the DMAIntrMask register are both set.
0	TxPktSent	0x0	This is set to '1' to record a TxPktSent interrupt when the TxPktSent bit in the DMATxStatus register and bit 0 of the DMAIntrMask register are both set.

Table 347 • CFG1

Bit Number	Name	Reset Value	Description
31	SOFT RESET	0x1	Setting this bit will put all modules within the PE-MCXMAC in the reset except the Host Interface.
30	SIMULATION RESET	0x0	Setting this bit will reset those registers, such as the random backoff timer, which are not controlled by the normal resets. (simulation only)
[29:20]	Reserved	0x0	Reserved

Table 347 • CFG1 (continued)

Bit Number	Name	Reset Value	Description
19	RESET RX MAC CONTROL	0x0	Setting this bit puts the PERMC Receive MAC control in reset.
18	RESET TX MAC CONTROL	0x0	Setting this bit puts the PETMC Transmit MAC control in reset.
17	RESET RX FUNCTION	0x0	Setting this bit puts the PERFN receive function block in reset. PERFN block performs the receive frame protocol.
16	RESET TX FUNCTION	0x0	Setting this bit puts the PETFN transmit function block in reset. PETFN block performs the frame transmission protocol.
[15:9]	Reserved	0x0	Reserved
8	LOOP BACK	0x0	Setting this bit causes the PETFN MAC transmit outputs to be looped back to the MAC receive inputs. Clearing this bit results in normal operation.
[7:6]	Reserved	0x0	Reserved
5	RECEIVE FLOW CONTROL ENABLE	0x0	Setting this bit causes the PERFN receive. MAC Control to detect and act on PAUSE flow control frames. Clearing this bit causes the receive MAC control to ignore PAUSE flow control frames.
4	TRANSMIT FLOW CONTROL ENABLE	0x0	Setting this bit allows the PETMC transmit. MAC Control to send PAUSE flow control frames when requested by the system. Clearing this bit prevents the transmit MAC control from sending flow control frames.
3	SYNCHRONIZE D RECEIVE ENABLE	0x0	This field is read only and indicates that the receive enable is synchronized to the receive stream.
2	RECEIVE ENABLE	0x0	Setting this bit allows the MAC to receive frames from the PHY. Clearing this bit prevents the reception of the frames.
1	SYNCHRONIZE D TRANSMIT ENABLE	0x0	This field is read only and indicates that the transmit enable is synchronized to the transmit stream.
0	TRANSMIT ENABLE	0x0	Setting this bit allows the MAC to transmit frames from the system. Clearing this bit prevents the transmission of the frames.

Table 348 • CFG2

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0x0	Reserved
[15:12]	PREAMBLE LENGTH	0x7	This field determines the length of the preamble field of the packet in bytes. Default is '0x7'.

Table 348 • CFG2 (continued)

Bit Number	Name	Reset Value	Description			
[9:8]	INTERFACE MODE	0x0	This field determines the type of interface the MAC is connected to. The Interface mode settings are as follows.			
			Interface mode		Bit 9	Bit 8
			Reserved		0	0
			Nibble mode (10/100 Mbps MII/RMII/SMII, ...)		0	1
			Byte mode (1000 Mbps GMII/TBI)		1	0
			Reserved		1	1
[7:6]	Reserved	0x0	Reserved			
5	HUGE FRAME ENABLE	0x0	Set this bit to allow frames longer than the MAXIMUM FRAME LENGTH to be transmitted and received. Clear this bit to have the MAC limits the length of frames at the MAXIMUM FRAME LENGTH value.			
4	LENGTH FIELD CHECKING	0x0	Set this bit to cause the MAC to check the frame's length field to ensure it matches the actual data field length. Clear this bit if no length field checking is desired.			
3	Reserved	0x0	Reserved			
2	PAD/CRC ENABLE	0x0	Set this bit to have the MAC pads all the short frames and appends a CRC to every frame whether or not padding is required. Clear this bit if frames, presented to the MAC, have a valid length and contain a CRC.			
1	CRC ENABLE	0x0	Set this bit to have the MAC appends a CRC to all the frames. Clear this bit if frames, presented to the MAC, have a valid length and contain a valid CRC. If the PAD/CRC ENABLE configuration bit or the per-packet PAD/CRC ENABLE is set, CRC ENABLE is ignored.			
0	FULL-DUPLEX	0x0	Setting this bit configures the PE-MCXMAC to operate in Full-duplex mode. Clearing this bit configures the PE-MCXMAC to operate in Half-duplex mode only.			

Table 349 • IFG

Bit Number	Name	Reset Value	Description
31	Reserved	0x0	Reserved
[30:24]	NON-BACK-TO-BACK INTER-PACKET-GAP PART 1 (IPGR1)	0x40	This programmable field represents the optional carrier Sense window, which is referenced in the IEEE 802.3/4.2.3.2.1 'Carrier Deference'. If a carrier is detected during the timing of IPGR1, the MAC defers to the carrier. However, the carrier becomes active after IPGR1; the MAC continues timing IPGR2 and transmit, knowingly causing a collision. This ensures fair access to the medium. The permitted range of values is 0x0 to IPGR2. Default is 0x40.
23	Reserved	0x0	Reserved
[22:16]	NON-BACK-TO-BACK INTER-PACKET-GAP PART 2 (IPGR2)	0x60	This programmable field represents the Non-Back-to-Back Inter-Packet-Gap in the bit times, which represent the minimum inter packet gap (IPG) of 96 bits.

Table 349 • IFG (continued)

Bit Number	Name	Reset Value	Description
[15:8]	MINIMUM IFG ENFORCEMENT	0x50	This programmable field represents the minimum size of management gap (IFG) to enforce between frames (expressed in bit times). A frame whose IFG is less than the programmed minimum IFG enforcement value is dropped. The default setting of 0x50 represents half of the nominal minimum IFG which is 160 bits.
7	Reserved	0x0	Reserved
[6:0]	BACK-TO-BACK INTER-PACKET-GAP	0x60	This programmable field represents the IPG between Back-to-Back packets (expressed in bit times). This is the IPG parameter used exclusively in full-duplex mode when two transmit packets are sent back-to-back. Set this field to the desired number of bits. The default setting of 0x60 represents the minimum IPG of 96 bits.

Table 350 • HALF_DUPLEX

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0x0	Reserved
[23:20]	ALTERNATE BINARY EXPONENTIAL BACKOFF TRUNCATION	0xA	This field is used when ALTERNATE BINARY EXPONENTIAL BACKOFF ENABLE is set. The value programmed is substituted for the Ethernet standard value of ten. Default is '0xA'.
19	ALTERNATE BINARY EXPONENTIAL BACKOFF ENABLE	0x0	Setting this bit configures the Tx MAC to use the ALTERNATE BINARY EXPONENTIAL BACKOFF TRUNCATION setting instead of the 802.3 standard tenth collision. Clearing this bit causes the Tx MAC to follow the standard binary exponential backoff rule.
18	BACKPRESSURE NO BACKOFF	0x0	Setting this bit configures the Tx MAC to immediately re-transmit following a collision during back pressure operation. Clearing this bit causes the Tx MAC to follow the binary exponential backoff rule.
17	NO BACKOFF	0x0	Setting this bit configures the Tx MAC to immediately re-transmit following a collision. Clearing this bit causes the Tx MAC to follow the binary exponential backoff rule.
16	EXCESSIVE DEFER	0x1	Setting this bit configures the Tx MAC to allow the transmission of a packet that has been excessively deferred. Clearing this bit causes the Tx MAC to abort the transmission of a packet that has been excessively deferred.
[15:12]	RETRANSMISSION MAXIMUM	0xF	This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The standard specifies the maximum number of attempts to be 0xF (15d).

Table 350 • HALF_DUPLEX (continued)

Bit Number	Name	Reset Value	Description
[11:10]	Reserved	0x0	Reserved
[9:0]	COLLISION WINDOW	0x37	This programmable field represents the slot time or collision window during which collisions might occur in a properly configured network. Since the collision window starts at the beginning of the transmission, the preamble and start frame delimiter (SFD) are included. The default of 0x37 (55d) corresponds to the count of the frame bytes at the end of the window.

Table 351 • MAX_FRAME_LENGTH

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0x0	Reserved
[15:0]	MAXIMUM FRAME LENGTH	0x600	This programmable field sets the maximum frame size in both the transmit and receive directions.

Table 352 • TEST

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0x0	Reserved
3	MAXIMUM FRAME LENGTH	0x0	Setting this bit causes the MAC to backoff for the maximum possible length of time. This test bit is used to predict the backoff times in Half-duplex mode.
2	REGISTERED TRANSMIT FLOW ENABLE	0x0	Registered transmit Half-duplex flow enable.
1	TEST PAUSE	0x0	Setting this bit allows the MAC to be paused via the host interface for the testing purposes.
0	SHORTCUT SLOT TIME	0x0	This bit allows the slot time counter to expire regardless of the current count. This bit is for the testing purposes only.

Table 353 • MII_CONFIG

Bit Number	Name	Reset Value	Description
31	RESET MII MGMT	0x0	Setting this bit resets MII Mgmt. Clearing this bit allows MII Mgmt to perform Mgmt read/write cycles as requested via the Host Interface.
30:6	Reserved	0x0	Reserved

Table 353 • MII_CONFIG (continued)

Bit Number	Name	Reset Value	Description
5	SCAN AUTO INCREMENT	0x0	Setting this bit causes MII Mgmt to continually read from a set of PHYs of contiguous address space. The starting address of the PHY is specified by the content of the PHY address field, which is recorded in the MII Mgmt Address register. Up to 31 PHY contiguous address space can be addressed. The last PHY, which is to be queried in this read sequence, is the one residing at address 0x31, after which the read sequence returns to the PHY, specified by the PHY address field.
4	PREAMBLE SUPPRESSION	0x0	Setting this bit causes MII Mgmt to suppress preamble generation and reduce the Mgmt cycle from 64 clocks to 32 clocks. This is in accordance with the IEEE 802.3/22.2.4.4.2. Clearing this bit causes MII Mgmt to perform Mgmt read/write cycles with the 64 clocks of preamble.
3	Reserved	0x0	Reserved
2:0	MGMT CLOCK SELECT	0x0	This field determines the clock frequency of the management data clock (MDC). MGMT Clock Select Encoding programming fields are given below.
			Mgmt Clock Select 2 1 0
			Source Clock divided by 4 0 0 0
			Source Clock divided by 4 0 0 1
			Source Clock divided by 6 0 1 0
			Source Clock divided by 8 0 1 1
			Source Clock divided by 10 1 0 0
			Source Clock divided by 14 1 0 1
			Source Clock divided by 20 1 1 0
			Source Clock divided by 28 1 1 1

Table 354 • MII_COMMAND

Bit Number	Name	Reset Value	Description
31:2	Reserved	0x0	Reserved
1	SCAN CYCLE	0x0	This bit causes MII Mgmt to perform Read cycles continuously. This is useful for monitoring Link Fail.
0	READ CYCLE	0x0	This bit causes MII Mgmt to perform a single Read cycle. The Read data is returned in Register 0xC (MII Mgmt Read Data).

Table 355 • MII_ADDRESS

Bit Number	Name	Reset Value	Description
[31:13]	Reserved	0x0	Reserved
[12:8]	PHY ADDRESS	0x00	This field represents the 5-bit PHY address field used in Mgmt cycles. Up to 31 PHYs can be addressed (0 is reserved).
[4:0]	REGISTER ADDRESS	0x00	This field represents the 5-bit register address field of Mgmt cycles. Up to 32 registers can be accessed.

Table 356 • MII_CTRL

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0x0	Reserved
[15:0]	MII MGMT CONTROL	0x0	When written, an MII Mgmt write cycle is performed using the 16-bit data and the pre-configured PHY and register addresses from the MII Mgmt Register (0x0A).

Table 357 • MII_STATUS

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0x0	Reserved
[15:0]	MII MGMT STATUS (PHY STATUS)	0x0	Following an MII Mgmt read cycle, the 16-bit data can be read from this location.

Table 358 • MII_INDICATORS

Bit Number	Name	Reset Value	Description
[31:3]	Reserved	0x0	Reserved
2	NOT VALID	0x0	When '1' is returned, this indicates MII Mgmt Read cycle is not completed and the Read Data is not yet validated.
1	SCANNING	0x0	When '1' is returned, this indicates a scan operation (continuous MII Mgmt Read cycles) is in progress.
0	BUSY	0x0	When '1' is returned, this indicates MII Mgmt block is currently performing an MII Mgmt read or write cycle.

Table 359 • INTERFACE_CTRL

Bit Number	Name	Reset Value	Description
31	RESET INTERFACE MODULE	0x0	Setting this bit resets the interface module. Clearing this bit allows for the normal operation. This bit can be used in the place of bits 23, 15, and 7, when just 1 interface module is connected.
[30:28]	Reserved	0x0	Reserved

Table 359 • INTERFACE_CTRL (continued)

Bit Number	Name	Reset Value	Description
27	TBIMODE	0x0	Setting this bit configures the A-RGMII module to expect TBI signals at the GMII interface. This bit should not be asserted unless this mode is being used.
26	GHDMODE	0x0	Setting this bit configures the A-RGMII module to expect half-duplex GMII at the GMII interface.
25	LHDMODE	0x0	Setting this bit configures the A-RGMII module to expect 10 or 100 half duplex MII at the GMII interface. This bit should not be asserted unless this mode is being used.
24	PHY MODE	0x0	Setting this bit configures the PESMII serial MII module to be in PHY mode. Link characteristics are taken directly from the Rx segments supplied by the PHY. Clearing this bit configures the PESMII to be in MAC to MAC mode. In this configuration, the Serial MII module reverts to the pre-defined settings of 100 Mbps, Full-duplex.
23	RESET PERMII	0x0	Setting this bit resets the PERMII module. Clearing this bit allows for normal operation.
[22:17]	Reserved	0x0	Reserved
16	SPEED	0x0	This bit configures the PERMII Reduced MII module with the current operating speed. When set, 100 Mbps mode is selected. When cleared, 10 Mbps mode is selected.
15	RESET PE100X	0x0	This bit resets the 4B/5B symbol encipher/decipher logic.
[14:11]	Reserved	0x0	Reserved
10	FORCE QUIET	0x0	When enabled, the transmit data is cleared which allows the contents of the cipher to be the output. When cleared, the normal operation is enabled. This affects the 4B/5B symbol encipher/decipher logic module only.
9	NO CIPHER	0x0	When enabled, the raw transmit 5B symbols are transmitted without ciphering. When disabled, the normal ciphering occurs. This affects the 4B/5B symbol encipher/decipher logic module only.
8	DISABLE LINK FAIL	0x0	When enabled, the 330 ms Link Fail timer is disabled, allowing shorter simulations. Removes the 330 ms link-up time before reception of streams is allowed. When cleared, normal operation occurs. Affects 4B/5B symbol encipher/decipher logic module only.
7	Reserved	0x0	Reserved
[6:1]	Reserved	0x0	Reserved
0	ENABLE JABBER PROTECTION	0x0	This bit enables the jabber protection logic. Jabber is the condition where a transmitter is stuck on for longer than 50ms preventing other stations from transmitting.

Table 360 • INTERFACE_STATUS

Bit Number	Name	Reset Value	Description
[31:11]	Reserved	0x0	Reserved
10	Reserved	0x0	Reserved
9	EXCESS DEFER	0x0	This bit sets when the MAC excessively defers a transmission. It clears when read. This bit latches high.
8	CLASH	0x0	When read as a '1', the Serial MII module is in the MAC to MAC mode with the partner in 10 Mbps and/or half-duplex mode indicative of a configuration error. When read as a '0', the Serial MII module is either in PHY mode or in a properly configured MAC to MAC mode.
7	JABBER	0x0	When read as a '1', the Serial MII PHY detects a jabber condition on the link. When read as a '0', the Serial MII PHY does not detect a jabber condition.
6	LINK OK	0x0	When read as a '1', the Serial MII PHY detects a valid link. When read as a '0', the Serial MII PHY does not detect a valid link.
5	FULL DUPLEX	0x0	When read as a '1', the Serial MII PHY operates in full-duplex mode. When read as a '0', the Serial MII PHY operates in half-duplex mode.
4	SPEED	0x0	When read as a '1', the Serial MII PHY operates at 100 Mbps mode. When read as a '0', the Serial MII PHY operates at 10 Mbps.
3	LINK FAIL	0x0	When read as a '1', the MII Management module reads the PHY link fail register to be 1'. When read as a '0', the MII Management module reads the PHY link fail register to be 0'. Note: For asynchronous host accesses, this bit must be read at least once every scan read cycle of the PHY.
2	Reserved	0x0	Reserved
1	Reserved	0x0	Reserved
0	Reserved	0x0	Reserved

Table 361 • STATION_ADDRESS1

Bit Number	Name	Reset Value	Description
[31:24]	STATION ADDRESS	0x0	This field holds the first octet of the station address.
[23:16]	STATION ADDRESS	0x0	This field holds the second octet of the station address.
[15:8]	STATION ADDRESS	0x0	This field holds the third octet of the station address.
[7:0]	STATION ADDRESS	0x0	This field holds the fourth octet of the station address.

Table 362 • STATION_ADDRESS2

Bit Number	Name	Reset Value	Description
[31:24]	STATION ADDRESS	0x0	This field holds the fifth octet of the station address.
[23:16]	STATION ADDRESS	0x0	This field holds the sixth octet of the station address.

Table 363 • FIFO_CFG0

Bit Number	Name	Reset Value	Description
[31:21]	Reserved	0x0	Reserved
20	fffenrply	0x0	This is a read only bit. When asserted, the FIFO transmit interface is enabled. When de-asserted, the FIFO transmit interface is disabled. The bit should be polled until it reaches the expected value.
19	stfenrply	0x0	This is a read only bit. When asserted, the FIFO PE-MCXMAC transmit interface module is enabled. When de-asserted, the FIFO PE-MCXMAC transmit interface module is disabled. The bit should be polled until it reaches the expected value.
18	frfenrply	0x0	This is a read only bit. When asserted, the FIFO receive interface module is enabled. When de-asserted, the FIFO receive interface module is disabled. The bit should be polled until it reaches the expected value.
17	srfernrply	0x0	This is a read only bit. When asserted, the FIFO PE-MCXMAC receive interface module is enabled. When de-asserted, the FIFO PE-MCXMAC receive interface module is disabled. The bit should be polled until it reaches the expected value.
16	wtmenrply	0x0	When asserted, the FIFO PE-MCXMAC watermark module is enabled. When de-asserted, the FIFO PE-MCXMAC watermark module is disabled. The bit should be polled until it reaches the expected value.
[15:13]	Reserved	0x0	Reserved
12	fffenreq	0x0	When asserted, this bit requests for enabling the FIFO fabric transmit interface module. When de-asserted, this bit requests for disabling the FIFO Fabric transmit interface module.
11	stfenreq	0x0	When asserted, this bit requests for enabling the FIFO PE-MCXMAC transmit interface module. When de-asserted, this bit requests for disabling the FIFO PE-MCXMAC transmit interface module.

Table 363 • FIFO_CFG0 (continued)

Bit Number	Name	Reset Value	Description
10	frfenreq	0x0	When asserted, this bit requests for enabling the FIFO fabric receive interface module. When de-asserted, this bit requests for disabling of the FIFO fabric receive interface module.
9	srfenreq	0x0	When asserted, this bit requests for enabling the FIFO PE-MCXMAC receive interface module. When de-asserted, this bit requests for disabling the FIFO PE-MCXMAC receive interface module.
8	wtmenreq	0x0	When asserted, this bit requests for enabling the FIFO PE-MCXMAC Watermark module. When de-asserted, this bit requests for disabling the FIFO PE-MCXMAC watermark module.
[7:5]	Reserved	0x0	Reserved
4	hstrstft	0x0	When asserted, this bit places the FIFO Fabric transmit interface module in reset.
3	hstrstst	0x0	When asserted, this bit places the FIFO PE-MCXMAC transmit interface module in reset.
2	hstrstfr	0x0	When asserted, this bit places the FIFO Fabric receive interface module in reset.
1	hstrstsr	0x0	When asserted, this bit places the FIFO PE-MCXMAC receive interface module in reset.
0	hstrstwt	0x0	When asserted, this bit places the FIFO PE-MCXMAC watermark module in reset.

Table 364 • FIFO_CFG1

Bit Number	Name	Reset Value	Description
[31:28]	Reserved	0x0	Reserved
[27:16]	cfgsrth[11:0]	0xFFF	This bit represents the minimum number of 4 byte locations, which are simultaneously stored in the receive RAM, relative to the beginning of the frame being input, before the fabric receive ready may be asserted.
[15:0]	cfgxoffrtx	0xFFFF	This bit represents the number of pause quanta after an XOFF pause frame is acknowledged, until the A-MCXFIFO re-asserts pause request if the A-MCXFIFO receive storage level remains higher than the low watermark.

Table 365 • FIFO_CFG2

Bit Number	Name	Reset Value	Description
[31:29]	Reserved	0x0	Reserved
[28:16]	cfghwm	0x1FFF	This bit represents the maximum number of 4 byte words that are simultaneously stored in the receive RAM before the transmit flow control enables and pause value facilitates an XOFF pause control frame.

Table 365 • FIFO_CFG2 (continued)

[15:13]	Reserved	0x0	Reserved
[12:0]	cfglwm	0x1FFF	This bit represents the minimum number of 4 byte words that are simultaneously stored in the receive RAM before transmit flow control enables and pause value facilitates an XON pause control frame in response to a previously transmitted XOFF pause control frame.

Table 366 • FIFO_CFG3

Bit Number	Name	Reset Value	Description
[31:28]	Reserved	0x0	Reserved
[27:16]	cfghwmft	0xFFF	This hex value represents the maximum number of 4 byte locations, which are simultaneously stored in the transmit RAM before the fthwm is asserted. The fthwm is asserted whenever the amount of four byte locations, used in the transmit FIFO data RAM, exceeds the value programmed in the cfghwmft host register.
[15:12]	Reserved	0x0	Reserved
[11:0]	cfgftth	0xFFF	This bit represents the minimum number of 4 byte locations which are simultaneously stored in the transmit RAM, relative to the beginning of the frame being input, before transmit packet start of frame is asserted.

Table 367 • FIFO_CFG4

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	hstfltrfrm	0x0	<p>These configuration bits are used to signal the drop frame conditions internal to the A-MCXFIFO.</p> <p>The bits correspond to the receive statistics vector input to A-MCXFIFO on a one per one basis. Receive statistics vector indicates the characteristics of the current receive frame and is input to A-MCXFIFO.</p> <p>The setting of this bits along with their don't care values in the hstfltrfrmdc not asserted, create the filter that drops the receive frame if the receive frame does not pass through the acceptable conditions.</p> <p>For example, if it is needed to drop a frame that contains a receive error, least significant fourth bit is set, and all receive frames that have frame receive error in receive statistics vector asserted are dropped.</p> <p>The hstfltrfrm bit and their corresponding receive statistics vector is as follows:</p> <p>Bit Description</p> <p>17: System Receive unicast Address.</p> <p>16: Truncated Frame.</p> <p>15: Receive long event.</p> <p>14: VLAN Tagged frame: frame's length/type field contained 0x8100 which is the VLAN protocol identifier.</p> <p>13: Frame was Unsupported Op-code.</p> <p>12: Frame was a PAUSE control frame</p> <p>11: Long event detected.</p> <p>10: Frame contained a dribble nibble.</p> <p>9: Broadcast address detected</p> <p>8: Multicast address detected</p> <p>7: Reception OK</p> <p>6: Length/Type field was neither a length nor type.</p> <p>5: Frame's length field out of range.</p> <p>4: Frame contained a CRC Error.</p> <p>3: Frame contained a code error.</p> <p>2: False carrier previously seen.</p> <p>1: RX_DV event previously seen.</p> <p>0: Whether or not a prior packet was dropped.</p>

Table 368 • FIFO_CFG5

Bit Number	Name	Reset Value	Description
[31:23]	Reserved	0x0	Reserved
22	cfghdplx	0x0	Assertion of this bit configures the A-MCXFIFO to enable the half duplex as a flow control mechanism. De-assertion of this bit configures the A-MCXFIFO to enable pause frames as a flow control mechanism.
21	srfull	0x0	Assertion of this read-only bit indicates that the maximum capacity of the receive FIFO storage is met or exceeded.

Table 368 • FIFO_CFG5 (continued)

20	hstsrfullclr	0x0	This bit should be written when it is desired to clear the srfull indicator bit. After the hstfullclr assertion, the srfull should be read until it becomes unasserted.
19	cfgbytmode	0x0	This bit should be asserted when the PE-MCXMAC is configured for GMII mode.
18	hstdrplt64	0x0	Setting this bit causes the frame to be dropped if a receive frame is less than 64 bytes in length.
[17:0]	hstfltrfrmdc	0X3FFFF	<p>These configuration bits indicate which receive statistics vectors are don't care for A-MCXFIFO frame drop circuitry. Receive statistics vector indicates the characteristics of the current receive frame.</p> <p>Setting of the hstfltrfrmdc bit, indicates a don't care for the receive statistics vector bit. These bits corresponds to receive statistics vector on a one per one basis. The hstfltrfrmdc bit and their corresponding receive statistics vector is as follows:</p> <p>BitDescription</p> <p>17: System Receive unicast Address</p> <p>16: Truncated Frame.</p> <p>15:Receive long event.</p> <p>14: VLAN Tagged frame: frame's length/type field contained 0x8100 which is the VLAN protocol identifier.</p> <p>13: Frame was Unsupported Op-code.</p> <p>12:Frame was a PAUSE control frame</p> <p>11:Long Event detected</p> <p>10:Frame contained a dribble nibble</p> <p>9:Broadcast address detected</p> <p>8:Multicast address detected</p> <p>7:Reception OK</p> <p>6:Length/Type field was neither a length nor type</p> <p>5:Frame's length field out of range.</p> <p>4:Frame contained a CRC Error.</p> <p>3:Frame contained a code error.</p> <p>2:False carrier previously seen</p> <p>1:RX_DV event previously seen</p> <p>0: Whether or not a prior packet was dropped</p>

Table 369 • FIFO_RAM_ACCESS0

Bit Number	Name	Reset Value	Description
31	hsttramwreq	0x0	Host transmit RAM write request
30	hsttramwack	0x0	Host transmit RAM write acknowledge
[29:24]	Reserved	0x0	Reserved
[23:16]	hsttramwdat [39:32]	0x0	<p>Host transmit RAM write data</p> <p>This is the upper byte of the transmit FIFO RAM data that is written at the address of hsttramwadx[10:0], if hsttramwadx[12] is negated and hsttramwreq is asserted.</p>
[15:13]	Reserved	0x0	Reserved
[12:0]	hsttramwadx	0x0	Host transmit RAM write address

Table 370 • FIFO_RAM_ACCESS1

Bit Number	Name	Reset Value	Description
[31:0]	hsttramwdat	0x0	Host transmit RAM write data

Table 371 • FIFO_RAM_ACCESS2

Bit Number	Name	Reset Value	Description
31	hsttramrreq	0x0	Host transmit RAM read request
30	hsttramrack	0x0	Host transmit RAM read acknowledge
[29:24]	Reserved	0x0	Reserved
[23:16]	hsttramrdat	0x0	Host transmit RAM read data. This is the upper byte of the transmit FIFO RAM data that was read at the address of the hsttramwadx[10:0], if the hsttramwadx[12] is negated and the hsttramwreq is asserted.
[15:13]	Reserved	0x0	Reserved
[12:0]	hsttramradx	0x0	Host transmit RAM read address

Table 372 • FIFO_RAM_ACCESS3

Bit Number	Name	Reset Value	Description
[31:0]	hsttramrdat	0x0	Host transmit RAM read data

Table 373 • FIFO_RAM_ACCESS4

Bit Number	Name	Reset Value	Description
31	hstramwreq	0x0	Host receive RAM write request
30	hstramwack	0x0	Host receive RAM write acknowledge
[29:24]	Reserved	0x0	Reserved
[23:16]	hstramwdat[39:32]	0x0	Host receive RAM write data This is the upper byte of the receive FIFO RAM data that is written at the address of the hstramwadx[11:0], if the hstramwadx[13] is negated and the hstramwreq is asserted.
[15:14]	Reserved	0x0	Reserved
[13:0]	hstramwadx	0x0	Host receive RAM write address

Table 374 • FIFO_RAM_ACCESS5

Bit Number	Name	Reset Value	Description
[31:0]	hstramwdat	0x0	Host receive RAM write data

Table 375 • FIFO_RAM_ACCESS6

Bit Number	Name	Reset Value	Description
31	hstramrreq	0x0	Host receive RAM read request
30	hstramrack	0x0	Host receive RAM read acknowledge
[29:24]	Reserved	0x0	Reserved
[23:16]	hstramrdat [39:32]	0x0	Host receive RAM read data This is the upper byte of the receive FIFO RAM data that is read at the address of the hstramwadx[10:0] if the hstramwadx[13] is negated and the hstramwreq is asserted.
[15:14]	Reserved	0x0	Reserved
[13:0]	hstramradx [13:0]	0x0	Host receive RAM read address

Table 376 • FIFO_RAM_ACCESS7

Bit Number	Name	Reset Value	Description
[31:0]	hstramrdat	0x0	Host receive RAM read data

Table 377 • TR64

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TR64	0x0	Transmit and receive 64 byte frame counter: Incremented for each good or bad transmitted and received frame, which is 64 bytes in length inclusive (excluding framing bits but including FCS bytes).

Table 378 • TR127

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TR127	0x0	Transmit and receive 65 to 127 byte frame counter: Incremented for each good or bad, transmitted and received frame, which is 65 to 127 bytes in length inclusive (excluding framing bits but including FCS bytes).

Table 379 • TR255

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TR255	0x0	Transmit and receive 128 to 255 byte frame counter: Incremented for each good or bad, transmitted and received frame, which is 128 to 255 bytes in length inclusive (excluding framing bits but including FCS bytes).

Table 380 • TR511

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TR511	0x0	Transmit and receive 256 to 511 byte frame counter: Incremented for each good or bad, transmitted and received frame, which is 256 to 511 bytes in length inclusive (excluding framing bits but including FCS bytes).

Table 381 • TR1K

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TR1K	0x0	Transmit and receive 512 to 1023 byte frame counter: Incremented for each good or bad, transmitted and received frame, which is 512 to 1023 bytes in length inclusive (excluding framing bits but including FCS bytes).

Table 382 • TRMAX

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TRMAX	0x0	Transmit and receive 1024 to 1518 byte frame counter: Incremented for each good or bad, transmitted and received frame, which is 1024 to 1518 bytes in length inclusive (excluding framing bits but including FCS bytes).

Table 383 • TRMGV

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TRMGV	0x0	Transmit and receive 1519 to 1522 byte VLAN frame counter: Incremented for each good virtual local area network (VLAN) transmitted and received frame which is 1519 to 1522 bytes in length inclusive (excluding framing bits but including FCS bytes).

Table 384 • RBYT

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0x0	Reserved
[23:0]	RBYT	0x0	Receive byte frame counter: The statistic counter register is incremented by the byte count of all the frames received, including those in the bad packets, excluding framing bits but including FCS bytes.

Table 385 • RPKT

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	RPKT	0x0	Receive packet counter: Incremented for each frame received packet (including bad packets, all the unicast, broadcast, and multicast packets).

Table 386 • RFCS

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RFCS	0x0	Receive FCS error counter: Incremented for each frame received that has an integral 64 to 1518 length and contains a FCS error.

Table 387 • RMCA

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	RMAC	0x0	Receive multicast packet counter: Incremented for each multicast good frame of lengths smaller than 1518 (non VLAN) or 1522 (VLAN) excluding the broadcast frames. This does not include range, length errors.

Table 388 • RBCA

Bit Number	Name	Reset Value	Description
[31:22]	Reserved	0x0	Reserved
[21:0]	RBCA	0x0	Receive broadcast packet counter: Incremented for each Broadcast good frame of lengths smaller than 1518 (non VLAN) or 1522 (VLAN) excluding the multicast frames. This does not look at range/length errors.

Table 389 • RXCF

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	RXCF	0x0	Receive control frame packet counter: Incremented for each MAC control frame received (PAUSE and Unsupported).

Table 390 • RXPF

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RXPF	0x0	Receive PAUSE frame packet counter: Incremented each time a valid PAUSE MAC control frame is received.

Table 391 • RXUO

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RXUO	0x0	Receive unknown OP code counter: Incremented each time a MAC control frame containing an op code other than a PAUSE is received.

Table 392 • RALN

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RALN	0x0	Receive alignment error counter: Incremented for each received frame from 64 to 1518. This contains an invalid FCS and is not an integral number of bytes.

Table 393 • RFLR

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0x0	Reserved
[15:0]	RFLR	0x0	Receive frame length error counter: Incremented for each frame received in which the 802.3 length field does not match with the number of the data bytes actually received (46-1500 bytes). The counter is not incremented if the length field is not a valid 802.3 length.

Table 394 • RCDE

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RCDE	0x0	Receive code error counter: Incremented each time a valid carrier is present and at least one invalid data symbol is detected.

Table 395 • RCSE

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RCSE	0x0	Receive false carrier counter: Incremented each time a false carrier is detected during idle, as defined by a 1 on RX_ER, and an '0xE' on RXD. The event is reported along with the statistics generated on the next received frame. Only one false carrier condition can be detected and logged between frames.

Table 396 • RUND

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RUND	0x0	Receive undersize packet counter: Incremented each time a frame is received which is less than 64 bytes in length and contains a valid FCS. This does not look at Range, Length errors.

Table 397 • ROVR

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	ROVR	0x0	Receive oversize packet counter: Incremented each time a frame is received which exceeded 1518 (non VLAN) or 1522 (VLAN) and contains a valid FCS. This does not look at Range, Length errors.

Table 398 • RFRG

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RFRG	0x0	Receive fragments counter: Incremented for each frame received which is less than 64 bytes in length and contains an invalid FCS. The received frame includes integral and non-integral lengths.

Table 399 • RJBR

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	Reserved	0x0	Receive jabber counter: Incremented for frames received which exceed 1518 (non VLAN) or 1522 (VLAN) bytes and contains an invalid FCS. The received frame, includes alignment errors.

Table 400 • RDRP

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	RDRP	0x0	Receive dropped packets counter: Incremented for frames received which are streamed to system but are later dropped due to lack of system resources.

Table 401 • TBYT

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0x0	Reserved
[23:0]	TPKT	0x0	Transmit byte counter: Incremented by the number of bytes that are transmitted including fragments of frames, which are involved in collisions. This count does not include preamble/SFD or jam bytes.

Table 402 • TPKT

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TPKT	0x0	Transmit packet counter: Incremented for each transmitted packet (including bad packets, excessive deferred packets, excessive collision packets, late collision packets, all unicast, broadcast, and multicast packets).

Table 403 • TMCA

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TPKT	0x0	Transmit packet counter: Incremented for each transmitted packet (including bad packets, excessive deferred packets, excessive collision packets, late collision packets, all unicast, broadcast, and multicast packets).

Table 404 • TBCA

Bit Number	Name	Reset Value	Description
[31:18]	Reserved	0x0	Reserved
[17:0]	TBCA	0x0	Transmit broadcast packet counter: Incremented for each broadcast frame transmitted (excluding multicast frames).

Table 405 • TXPF

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TXPF	0x0	Transmit PAUSE frame packet counter: Incremented each time a valid PAUSE MAC Control frame is transmitted.

Table 406 • TDFR

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TDFR	0x0	Transmit deferral packet counter: Incremented for each frame, which is deferred on its first transmission attempt. This does not include frames involved in collisions.

Table 407 • TEDF

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TEDF	0x0	Transmit excessive deferral packet counter: Incremented for aborted frames which are deferred for an excessive period of time (3036 byte times).

Table 408 • TSCL

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TSCL	0x0	Transmit single collision packet counter: Incremented for each transmitted frame which experiences exactly one collision during the transmission.

Table 409 • TMCL

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TMCL	0x0	Transmit multiple collision packet counter: Incremented for each transmitted frame which experiences 2-15 collisions (including any late collisions).

Table 410 • TLCL

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TLCL	0x0	Transmit late collision packet counter: Incremented for each transmitted frame which experiences a late collision during a transmission attempt.

Table 411 • TXCL

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TXCL	0x0	Transmit excessive collision packet counter: Incremented for each frame that experiences 16 collisions during the transmission and is aborted.

Table 412 • TNCL

Bit Number	Name	Reset Value	Description
[31:13]	Reserved	0x0	Reserved
[12:0]	TNCL	0x0	Transmit total collision counter: Incremented by the number of collisions experienced during the transmission of a frame due to simultaneous transmitting and receiving.

Table 413 • TPFH

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TPFH	0x0	Transmit PAUSE frames honored counter: Incremented each time a valid PAUSE MAC control frame is transmitted and honored.

Table 414 • TDRP

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TDRP	0x0	Transmit drop frame counter: Incremented each time the transmit PAUSE frame honored input to PE-MSTAT is asserted.

Table 415 • TJBR

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TJBR	0x0	Transmit jabber frame counter: Incremented for each oversized transmitted frame with an incorrect FCS value.

Table 416 • TFCS

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TFCS	0x0	Transmit FCS error counter: incremented for each valid sized packet with an incorrect FCS value.

Table 417 • TXCF

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TXCS	0x0	Transmit control frame counter: incremented for each valid size frame with a type field signifying a control frame.

Table 418 • TOVR

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TOVR	0x0	Transmit oversize frame counter: incremented for each oversized transmitted frame with a correct FCS value.

Table 419 • TUND

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TUND	0x0	Transmit undersize frame counter: Incremented for each frame less than 64 bytes, with a correct FCS value.

Table 420 • TFRG

Bit Number	Name	Reset Value	Description
[31:12]	Reserved	0x0	Reserved
[11:0]	TFRG	0x0	Transmit fragment counter: incremented for each frame less than 64 bytes, with a incorrect FCS value.

Table 421 • CAR1

Bit Number	Name	Reset Value	Description
31	C164	0x0	Carry register 1 TR64 counter carry bit
30	C1127	0x0	Carry register 1 TR127 counter carry bit
29	C1255	0x0	Carry register 1 TR255 counter carry bit
28	C1511	0x0	Carry register 1 TR511 counter carry bit
27	C11k	0x0	Carry register 1 TR1K counter carry bit
26	C1MAX	0x0	Carry register 1 TRMAX counter carry bit
25	C1MGV	0x0	Carry register 1 TRMGV counter carry bit
[24:17]	Reserved	0x0	Reserved
16	C1RBY	0x0	Carry register 1 RBYT counter carry bit
15	C1RPK	0x0	Carry register 1 RPKT counter carry bit
14	C1RFC	0x0	Carry register 1 RFCS counter carry bit
13	C1RMC	0x0	Carry register 1 RMCA counter carry bit
12	C1RBC	0x0	Carry register 1 RBCA counter carry bit
11	C1RXC	0x0	Carry register 1 RXCF counter carry bit
10	C1RXP	0x0	Carry register 1 RXPF counter carry bit
9	C1RXU	0x0	Carry register 1 RXUO counter carry bit
8	C1RAL	0x0	Carry register 1 RALN counter carry bit
7	C1RFL	0x0	Carry register 1 RFLR counter carry bit

Table 421 • CAR1 (continued)

6	C1RCD	0x0	Carry register 1 RCDE counter carry bit
5	C1RCS	0x0	Carry register 1 RCSE counter carry bit
4	C1RUN	0x0	Carry register 1 RUND counter carry bit
3	C1ROV	0x0	Carry register 1 ROVR counter carry bit
2	C1RFR	0x0	Carry register 1 RFRG counter carry bit
1	C1RJB	0x0	Carry register 1 RJBR counter carry bit
0	C1RDR	0x0	Carry register 1 RDRP counter carry bit

Table 422 • CAR2

Bit Number	Name	Reset Value	Description
[31:20]	Reserved	0x0	Reserved
19	C2TJB	0x0	Carry register 2 TJBR counter carry bit
18	C2TFC	0x0	Carry register 2 TXFC counter carry bit
17	C2TCF	0x0	Carry register 2 TXCF counter carry bit
16	C2TOV	0x0	Carry register 2 TOVR counter carry bit
15	C2TUN	0x0	Carry register 2 TUND counter carry bit
14	C2TFG	0x0	Carry register 2 TFRG counter carry bit
13	C2TBY	0x0	Carry register 2 TBYT counter carry bit
12	C2TPK	0x0	Carry register 2 TPKT counter carry bit
11	C2TMC	0x0	Carry register 2 TMCA counter carry bit
10	C2TBC	0x0	Carry register 2 TBCA counter carry bit
9	C2TPF	0x0	Carry register 2 TXPF counter carry bit
8	C2TDF	0x0	Carry register 2 TDFR counter carry bit
7	C2TED	0x0	Carry register 2 TEDF counter carry bit
6	C2TSC	0x0	Carry register 2 TSCL counter carry bit
5	C2TMA	0x0	Carry register 2 TMCL counter carry bit
4	C2TLC	0x0	Carry register 2 TLCL counter carry bit
3	C2TXC	0x0	Carry register 2 TXCL counter carry bit
2	C2TNC	0x0	Carry register 2 TNCL counter carry bit
1	C2TPH	0x0	Carry register 2 TPFH counter carry bit
0	C2TDP	0x0	Carry register 2 TDRP counter carry bit

Table 423 • CAM1

Bit Number	Name	Reset Value	Description
31	M164	0x1	Mask register 1 TR64 counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
30	M1127	0x1	Mask register 1 TR127 counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
29	M1255	0x1	Mask register 1 TR255 counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
28	M1511	0x1	Mask register 1 TR511 counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
27	M11k	0x1	Mask register 1 TR1K counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
26	M1MAX	0x1	Mask register 1 TRMAX counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
25	M1MGV	0x1	Mask register 1 TRMGV counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
[24:17]	Reserved	0x0	Reserved
16	M1RBY	0x1	Mask register 1 RBYT counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
15	M1RPK	0x1	Mask register 1 RPKT counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
14	M1RFC	0x1	Mask register 1 RFCS counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
13	M1RMC	0x1	Mask register 1 RMCA counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
12	M1RBC	0x1	Mask register 1 RBCA counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
11	M1RXC	0x1	Mask register 1 RXCF counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
10	M1RXP	0x1	Mask register 1 RXPf counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
9	M1RXU	0x1	Mask register 1 RXUO counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit

Table 423 • CAM1 (continued)

Bit Number	Name	Reset Value	Description
8	M1RAL	0x1	Mask register 1 RALN counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
7	M1RFL	0x1	Mask register 1 RFLR counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
6	M1RCD	0x1	Mask register 1 RCDE counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
5	M1RCS	0x1	Mask register 1 RCSE counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
4	M1 RUN	0x1	Mask register 1 RUND counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
3	M1ROV	0x1	Mask register 1 ROVR counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
2	M1RFR	0x1	Mask register 1 RFRG counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
1	M1RJB	0x1	Mask register 1 RJBR counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
0	M1RDR	0x1	Mask register 1 RDRP counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit

Table 424 • CAM2

Bit Number	Name	Reset Value	Description
[31:20]	Reserved	0x0	Reserved
19	M2TJB	0x1	Mask register 2 TJBR counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
18	M2TFC	0x1	Mask register 2 TXFC counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
17	M2TCF	0x1	Mask register 2 TXCF counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
16	M2TOV	0x1	Mask register 2 TOVR counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
15	M2TUN	0x1	Mask register 2 TUND counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit

Table 424 • CAM2 (continued)

Bit Number	Name	Reset Value	Description
14	M2TFG	0x1	Mask register 2 TFRG counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
13	M2TBY	0x1	Mask register 2 TBYT counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
12	M2TPK	0x1	Mask register 2 TPKT counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
11	M2TMC	0x1	Mask register 2 TMCA counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
10	M2TBC	0x1	Mask register 2 TBCA counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
9	M2TPF	0x1	Mask register 2 TXPF counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
8	M2TDF	0x1	Mask register 2 TDFR counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
7	M2TED	0x1	Mask register 2 TEDF counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
6	M2TSC	0x1	Mask register 2 TSCL counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
5	M2TMA	0x1	Mask register 2 TMCL counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
4	M2TLC	0x1	Mask register 2 TLCL counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
3	M2TXC	0x1	Mask register 2 TXCL counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
2	M2TNC	0x1	Mask register 2 TNCL counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
1	M2TPH	0x1	Mask register 2 TPFH counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit
0	M2TDP	0x1	Mask register 2 TDRP counter carry bit 0: Unmask the counter carry bit 1: Mask the counter carry bit

Table 425 • SGMII CONTROL

Bit Number	Name	Reset Value	Description
15	PHY RESET	0x0	Setting this bit causes the PETEX, PEREX, and PEANX sub-modules in the M-SGMII core to be reset. This bit is self-clearing.
14	LOOP BACK	0x0	Setting this bit causes the M-SGMII loopback. Clearing this bit results in normal operation.
13	Reserved	0x0	Reserved.
12	AUTO-NEGOTIATION ENABLE	0x0	Setting this bit enables the auto-negotiation process.
[11:10]	Reserved	0x0	Reserved.
9	RESET AUTO-NEGOTIATION	0x0	Setting this bit causes the auto-negotiation process to restart. This action is only available when auto-negotiation has been enabled.
[8:0]	Reserved	0x0	Reserved

Table 426 • SGMII STATUS

Bit Number	Name	Reset Value	Description
[15:9]	Reserved	0x0	Reserved.
8	EXTENDED STATUS	0x0	This bit returns '1' on read to indicate that the PHY status information is also contained in EXTENDED STATUS register.
7	Reserved	0x0	Reserved.
6	MF PREAMBLE SUPPRESSION ENABLE	0x0	This bit indicates whether the PHY is capable of handling MII management frames without the 32-bit preamble field. Returns '1' on read to indicate the support for suppressed preamble MII management frames.
5	AUTO-NEGOTIATION COMPLETE	0x0	This bit indicates that the Auto-negotiation process is completed. Returns '0' on read when either the auto-negotiation process is underway or when the Auto-negotiation function is disabled.
4	REMOTE FAULT	0x0	This bit returns '1' on read to indicate a remote fault condition has been detected between the M-SGMII and the PHY. This bit latches high in order for software to detect the condition. Each read of the STATUS register clears this bit.
3	AUTO-NEGOTIATION ABILITY	0x0	When '1', this bit indicates that the M-SGMII has the ability to perform auto-negotiation. Returns '1' on read.
2	LINK STATUS		This bit indicates that a valid link is established between the M-SGMII and the PHY. Returns '0' on read to indicate that there is no valid link is established. This bit latches low to allow software polling to detect a failure condition.
1	Reserved	0x0	Reserved
0	EXTENDED CAPABILITY	0x1	This bit returns '1' on read to indicates that the M-SGMII contains the extended set of registers (those beyond CONTROL and STATUS).

Table 427 • AN SGMII ADVERTISEMENT

Bit Number	Name	Reset Value	Description
15	LINK UP	0x0	Assertion of this bit indicates that the link between M-SGMII and PHY is up.
[14:13]	Reserved	0x0	Reserved.
12	FULL DUPLEX	0x0	Assertion of this bit indicates that the link between M-SGMII and PHY is up and transferring data in Full-duplex mode.
[11:10]	LINK SPEED	0x0	Assertion of these 2 bits indicate that the link between M-SGMII and PHY is up and the speed that the link is transferring data is as mentioned below:
			LINK SPEED [11] LINK SPEED [10] Capability
			1 1 Reserved
			1 0 1000 Mbps
			0 1 100 Mbps
			0 0 10 Mbps
[9:0]	Reserved	0x0	These bits must always be written '0000000001' for correct M-SGMII operation.

Table 428 • AN LINK PARTNER BASE PAGE ABILITY

Bit Number	Name	Reset Value	Description
15	LINK UP	0x0	When the M-SGMII is integrated to a MAC, such as the PE-MCXMAC, and is communicating with another SGMII PHY module, assertion of this bit indicates that the link is up. When the M-SGMII is integrated to a PHY and is not integrated to MAC, this bit is invalid.
[14:13]	Reserved	0x0	Reserved.
12	FULL DUPLEX	0x0	Assertion of this bit indicates that LINK UP bit of the AN SGMII partner base page ability register is asserted and the link is transferring data in full-duplex mode.
[11:10]	LINK SPEED	0x0	Indicates the speed of the link as mentioned below when LINK UP bit of the AN SGMII partner base page ability register is asserted.
			LINK SPEED [11] LINK SPEED [10] Capability
			1 1 Reserved
			1 0 1000 Mbps
			0 1 100 Mbps
			0 0 10 Mbps
[9:0]	Reserved	0x0	Reserved

Table 429 • AN EXPANSION

Bit Number	Name	Reset Value	Description
[15:3]	Reserved	0x0	Reserved.
2	NEXT PAGE ABLE	0x0	Returns '1' on read to indicate that the local device supports the next page function.
1	PAGE RECEIVED	0x0	Returns '1' on read to indicates that a new page has been received and stored in the applicable AN LINK PARTNER BASE PAGE ABILITY or AN NEXT PAGE TRANSMIT register. This bit latches High for detection by software when polling. The bit is cleared on a read to the register.
0	Reserved	0x0	Reserved.

Table 430 • AN NEXT PAGE TRANSMIT

Bit Number	Name	Reset Value	Description
15	NEXT PAGE	0x0	Assert this bit to indicate additional Next Pages to follow. Clear the bit to indicate the last page.
14	Reserved	0x0	Reserved
13	MESSAGE PAGE	0x0	Assert this bit to indicate a Message Page. Clear the bit to indicate an Unformatted Page.
12	ACKNOWLEDGE 2	0x0	Used by the Next Page function to indicate that the device has the ability to comply with the message. Assert this bit if the local device will comply with the message. Clear the bit if the local device cannot comply with message.
11	TOGGLE	0x0	This bit is read only. Used to ensure synchronization with the Link Partner during Next Page exchange. This bit always takes the opposite value to the Toggle bit of the previously exchanged Link Code Word. The initial value in the first Next Page transmitted is the inverse of bit 11 in the base Link Code Word.
[10:0]	MESSAGE / UNFORMATTED CODE FIELD	0x0	Message pages are formatted pages that carry a predefined Message Code, which is enumerated in IEEE 802.3u/Annex 28C. Unformatted code fields take an arbitrary value.

Table 431 • AN NEXT PAGE TRANSMIT

Bit Number	Name	Reset Value	Description
15	NEXT PAGE		The Link Partner asserts this bit to indicate additional Next Pages to follow. When '0', indicates last Next Page from link partner.
14	Reserved		Reserved
13	MESSAGE PAGE		When '1', indicates Message Page. When '0', indicates Unformatted Page.

Table 431 • AN NEXT PAGE TRANSMIT (continued)

12	ACKNOWLEDGE 2		Indicates link partner's ability to comply with the message. When '1', Link Partner complies with message. When '0', link partner cannot comply with message.
11	TOGGLE		Used to ensure synchronization with the link partner during next page exchange. This bit always takes the opposite value to the Toggle bit of the previously exchanged link code word. The initial value in the first next page transmitted is the inverse of bit 11 in the base link code word.
[10:0]	MESSAGE / UNFORMATTED CODE FIELD		Message pages are formatted pages that carry a predefined message code, which is enumerated in the IEEE 802.3u/Annex 28C. Unformatted code fields take an arbitrary value.

Table 432 • EXTENDED STATUS

Bit Number	Name	Reset Value	Description
15	1000BASE-X FULL-DUPLEX	0x1	When '1', indicates that the PHY can operate in 1000BASE-X Full-duplex mode. When '0', indicates that the PHY cannot operate in 1000BASE-X Full-duplex mode.
14	1000BASE-X HALF-DUPLEX	0x0	When '1', indicates that the PHY can operate in 1000BASE-X Half-duplex mode. When '0', indicates that the PHY cannot operate in 1000BASE-X Half-duplex mode.
13	1000BASE-T FULL-DUPLEX	0x1	When '1', indicates that the PHY can operate in 1000BASE-T Full-duplex mode. When '0', indicates that the PHY cannot operate in 1000BASE-T Full-duplex mode. Returns '1' on read.
12	1000BASE-T HALF-DUPLEX	0x0	When '1', indicates that the PHY can operate in 1000BASE-T Half-duplex mode. When '0', indicates PHY cannot operate in 1000BASE-T Half-duplex mode.
[11:0]	Reserved	0x0	Reserved.

Table 433 • JITTER DIAGNOSTICS

Bit Number	Name	Reset Value	Description			
15	JITTER DIAGNOSTIC ENABLE	0x0	Set this bit to enable the M-SGMII to transmit the jitter test patterns defined in IEEE 802.3z 36A. Clear this bit to enable normal transmit operation.			
[14:12]	JITTER PATTERN SELECT	0x0	Selects the jitter pattern to be transmitted in Diagnostics mode. Encoding of this field is shown in the following table:			
			Jitter Pattern Select	Bit 14	Bit 13	Bit 12
			User defined custom pattern	0	0	0
			Annex 36A defined high frequency	0	0	1
			Annex 36A defined mixed frequency	0	1	0
			Custom defined low frequency	0	1	1
			Random jitter pattern	1	0	0
			Annex 36A defined low frequency	1	0	1
			Reserved	1	1	0
			Reserved	1	1	1
[11:10]	Reserved	0x0	Reserved			
[9:0]	CUSTOM JITTER PATTERN	0x0	Used in conjunction with JITTER PATTERN SELECT and JITTER DIAGNOSTIC ENABLE. Set this field to the desired custom pattern, which is transmitted continuously.			

Table 434 • TBI CONTROL

Bit Number	Name	Reset Value	Description
15	SOFT RESET	0x0	This bit resets the functional modules in the M-SGMII. Clear it for normal operation. Its default is '0'.
14	SHORTCUT LINK TIMER	0x0	Set this bit to reduce the amount of simulation time needed to time the 1.6ms Link Timer. Clear it for normal operation.
13	DISABLE RECEIVE RUNNING DISPARITY	0x0	Set this bit to disable the running disparity calculation and checking in the receive direction. This bit must be '0' for correct M-SGMII operation.
12	DISABLE TRANSMIT RUNNING DISPARITY	0x0	Set this bit to disable the running disparity calculation and checking in the transmit direction. This bit must be '0' for correct M-SGMI operation.
[11:9]	Reserved	0x0	Reserved

Table 434 • TBI CONTROL (continued)

8	AUTO-NEGOTIATION SENSE	0x0	Set this bit to allow the auto-negotiation for 1000BASE-X, which is used to exchange information between link partners. Clear this bit when IEEE 802.3z Clause 37 behavior is desired, which results in the link not coming up.
[7:6]	Reserved	0x0	Reserved.
5	RECEIVE CLOCK SELECT	0x0	Set this bit to configure the M-SGMII to accept a 125 MHz receive clock from the SERDES PHY. Clear this bit to allow the M-SGMII to accept dual split-phase 62.5 MHz receive clocks. This bit must be '0' for correct M-SGMII operation.
4	GMII MODE	0x0	When cleared, this bit defines the M-SGMII as being in 1000BASE-X SERDES mode. This bit must be '0' for correct M-SGMII operation.
[3:2]	Reserved	0x0	Reserved
1	ENABLE WRAP	0x0	Set this bit to configure the SERDES in Loopback mode. Clear this bit to permit normal operation.
0	ENABLE COMMA DETECT	0x0	Set this bit to allow the SERDES PHY to perform code group alignment based upon the detection of a comma.

11.9 CoreMACFilter Overview

CoreMACFilter provides a solution for SmartFusion2 integrated media access control (MAC) address filtering. The core provides an external filtering mechanism based on unicast (UCAD), multicast (MCAD), and broadcast (BCAD) flags. It implements the desired mechanism to pass the frames to upper layer. The upper layer determines to reject or accept the frames.

The CoreMACFilter filters the unwanted frames based on the following:

- Local base station MAC address
- MCAD
- BCAD
- Hash-unicast
- Hash-multicast of filter operating modes

A 128-bit hash table is used for hash-unicast and hash-multicast frame filtering. The frame filtering is performed on the destination MAC address of the received frame. The CoreMACFilter has an APB interface to allow the address filtering configurations and other MAC configurations.

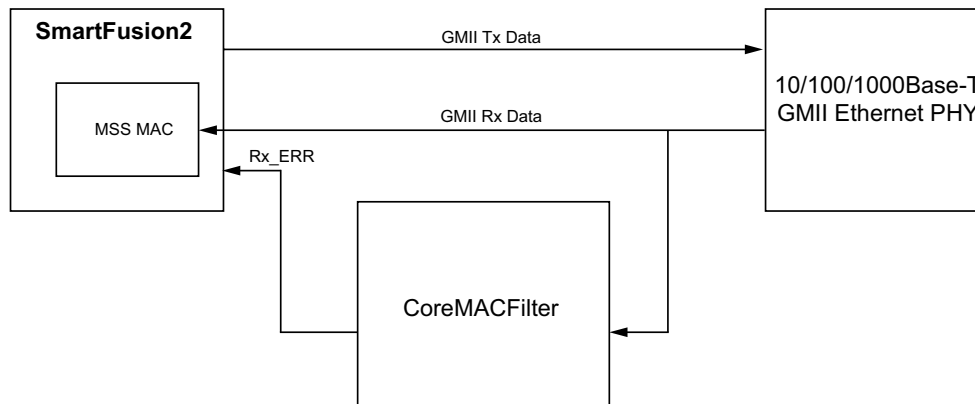
11.9.1 Features

CoreMACFilter supports the following:

- Provides an advanced peripheral bus (APB) interface for control and status register access
- Supports UCAD, MCAD, and broadcast type of packets
- Supports hash based address filtering for UCAD and MCAD packets
- Provides mechanism to the upper layer to reject or accept the frames

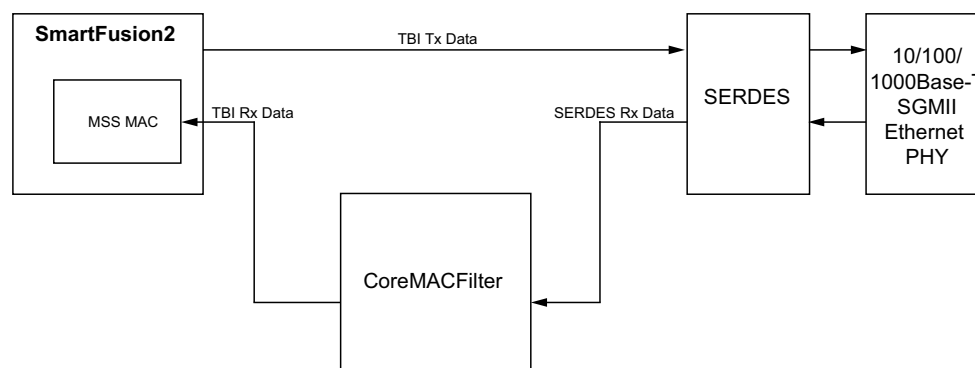
The following figure shows the CoreMACFilter interaction with MSS MAC and GMII Ethernet PHY.

Figure 164 • CoreMACFilter Interaction with MSS MAC and GMII Ethernet PHY



The following figure shows CoreMACFilter interaction with MSS MAC and SGMII Ethernet PHY.

Figure 165 • CoreMACFilter interaction with MSS MAC and SGMII Ethernet PHY



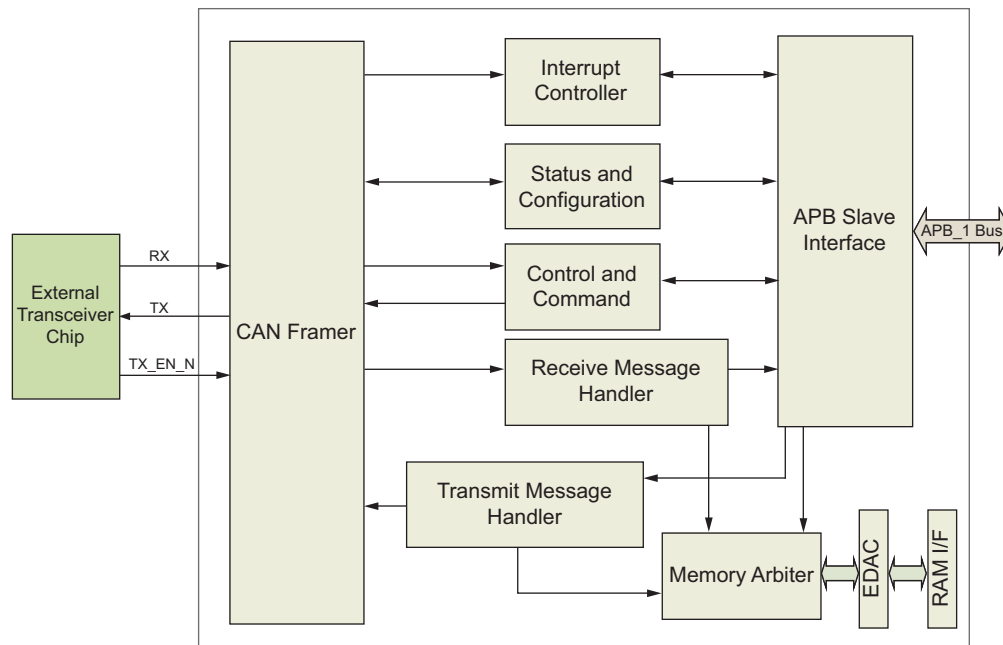
For more information on CoreMACFilter, refer to CoreMACFilter handbook.

12 CAN Controller

SmartFusion2 SoC FPGAs contain an integrated control area network (CAN) peripheral. The CAN controller is an advanced peripheral bus (APB_1) slave on the MSS AHB bus matrix. Refer to the [AHB Bus Matrix](#), page 210 for a detailed description. A master such as the Cortex-M3 processor or a master in the FPGA fabric configures the CAN controller through the APB slave.

The CAN controller in the SmartFusion2 device supports the concept of mailboxes. It is compliant to the international CAN standard defined in ISO 11898-1. It contains 32 receive buffers. Each buffer has its own message filter and 32 transmit buffers with prioritized arbitration scheme. For optimal support of higher-layer protocols (HLP) such as DeviceNet, the message filter also covers the first two data bytes of the message payload. A block diagram of the CAN controller is shown in the following figure. Transmit and receive message buffers are single error corrected, double error detected (SECDED) through the error detection and correction (EDAC) controller. The functional behavior of the CAN instance must be defined at the application level using the SmartFusion2 MSS CAN firmware driver provided by Microsemi. Refer to the **CAN Firmware Driver User Guide** for more details.

Figure 166 • CAN Controller Block Diagram



12.1 Features

12.1.1 Compliance

- Full CAN 2.0A and 2.0B compliant
- Conforms to ISO 11898-1

12.1.2 Receive Path

- 32 receive buffers
- Each buffer has its own message filter
- Message filter covers: ID, IDE, remote transmission request (RTR), data byte 1, and data byte 2
- Message buffers can be linked together to build a bigger message array.
- Automatic RTR response handler with optional generation of RTR interrupt

12.1.3 Transmit Path

- 32 transmit message holding registers with programmable priority arbitration
- Message abort command
- Single-shot transmission (SST); no automatic retransmission upon error or arbitration loss

12.1.4 EDAC

An internal 256 x 32 RAM in the CAN controller is EDAC protected. EDAC configurations and error counters related to the CAN are maintained in MSS system registers. For example, to enable or disable the EDAC component of the CAN, set the CAN_EDAC_EN bit (6th bit in the SYSREG EDAC_CR register) to 1. By default, EDAC is disabled (CAN_EDAC_EN is set to 0). Refer to [Table 439](#), page 451 for more information.

After power-up, the internal SRAM is not initialized and any READ to the memory location would result in an ECC error if EDAC is enabled. To initialize the SRAM, you can put the CAN controller into SRAM Test mode, initialize the SRAM, and enable the EDAC. If SECDDED is enabled, Microsemi recommends that the CAN controller be put into SRAM Test mode and the RAM initialized with user defined known data before operation so that a future read or an uninitialized address does not trigger a SECDDED error. Refer to the [Use Cases](#), page 448 for more details on how to put the CAN controller into SRAM Test mode.

12.1.5 Enable or Disable Control

The CAN controller can be enabled or disabled using the MSS configurator in Libero SoC, depending on the application needs. When it is disabled, the CAN controller is held in reset (lowest power state). Refer to the [Hardware Design Flow](#), page 446 for more information on how to enable or disable the CAN controller using Libero SoC.

12.1.6 System Dependencies

12.1.6.1 Reset

The CAN controller resets to zero on power-up and is held in reset until enabled, as shown in [CAN Controller Soft Reset Bit in the SOFT_RESET_CR Register](#), page 452 in the SOFT_RESET_CR register. The CAN controller can be reset by writing to CAN_SOFTRESET (bit13) of the SOFT_RESET_CR register. The SOFT_RESET_CR register is located in the SYSREG block. Refer to the [System Register Block](#), page 670 for more details.

12.2 Functional Description

12.2.1 CAN Controller Interface Signals

This section describes the CAN bus interfaces. The external interface signals connecting the SmartFusion2 device to an off-chip CAN transceiver are listed in the following table.

Table 435 • CAN BUS Interface

Signal Name	Direction	Description
RX	Input	CAN bus receive signal. This signal connects to the receiver bus of the external driver.
TX	Output	CAN bus transmit signal. This signal connects to the external driver.
TX_EN_N	Output	External driver enable control signal This signal is used to enable or disable an external CAN transceiver. TX_EN_N is asserted when the CAN controller is stopped or if the CAN state is bus-off (shut down completely).

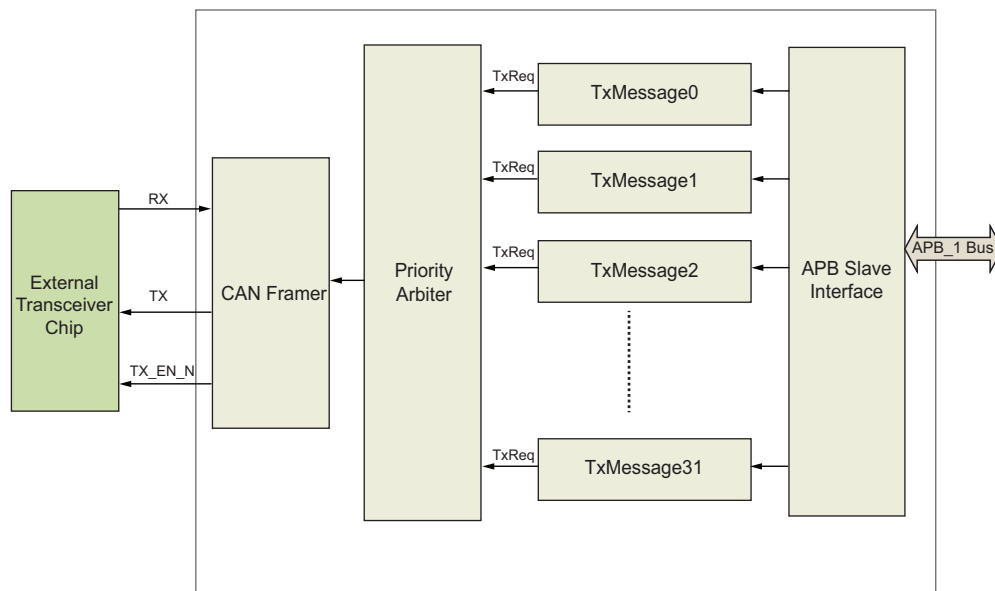
When enabled, CAN ports are configured to connect to SmartFusion2 multi-standard I/Os (MSIOs) by default. CAN signals can also be configured to interface with the FPGA fabric and the MSS general purpose inputs/outputs (GPIOs). The CAN configurator within Libero SoC allows selection from among the fabric, MSIOs, and GPIOs.

Note: The MSIOs allocated to the CAN instance are shared with other MSS peripherals. These shared I/Os are available to connect to the MSS GPIOs and other peripherals when the CAN instance is disabled or if the CAN instance ports are only connected to the FPGA fabric. Refer to the [MSS CAN Configurator User Guide](#) for more details.

12.2.2 Transmit Procedures

The CAN controller provides 32 transmit message holding buffers. An internal priority arbiter selects the message according to the chosen arbitration scheme. Upon transmission of a message or message arbitration loss, the priority arbiter re-evaluates the message priority of the next message. The following figure gives an overall view of the transmit message buffers.

Figure 167 • Transmit Message Buffers



Two types of message priority arbitration are supported. The type of arbitration is selected using the configuration register. Following are the arbitration types:

- **Round Robin:** Buffers are served in a defined order: 0-1-2... 31-0-1... A particular buffer is only selected if its TxReq flag is set. This scheme guarantees that all buffers receive the same probability to send a message.
- **Fixed Priority:** Buffer 0 has the highest priority. This way it is possible to designate buffer 0 as the buffer for error messages and it is guaranteed that they are sent first.

Note: RTR message requests are served before transmit message buffers are handled. For example, RTRreq0, RTRreq31, TxMessage0, TxMessage1, and TxMessage31.

12.2.2.1 Procedure for Sending a Message

1. Write message into an empty transmit message holding buffer. An empty buffer is indicated by the TxReq that is equal to zero.
2. Request transmission by setting the respective TxReq flag to 1.
3. The TxReq flag remains set as long as the message transmit request is pending. The content of the message buffer must not be changed while the TxReq flag is set.
4. The internal message priority arbiter selects the message according to the chosen arbitration scheme.
5. Once the message is transmitted, the TxReq flag is set to zero and the tx_msg interrupt status bit is asserted

12.2.2.2 Remove a Message from a Transmit Holding Register

A message can be removed from the transmit holding buffer by asserting the TxAbort flag. The content of a particular transmit message buffer can be removed by setting TxAbort to 1 to request message removal. This flag remains set as long as the message abort request is pending. It is cleared when either the message wins arbitration (tx_msg interrupt active) or the message is removed (tx_msg interrupt inactive).

12.2.2.2.1 Single-Shot Transmission

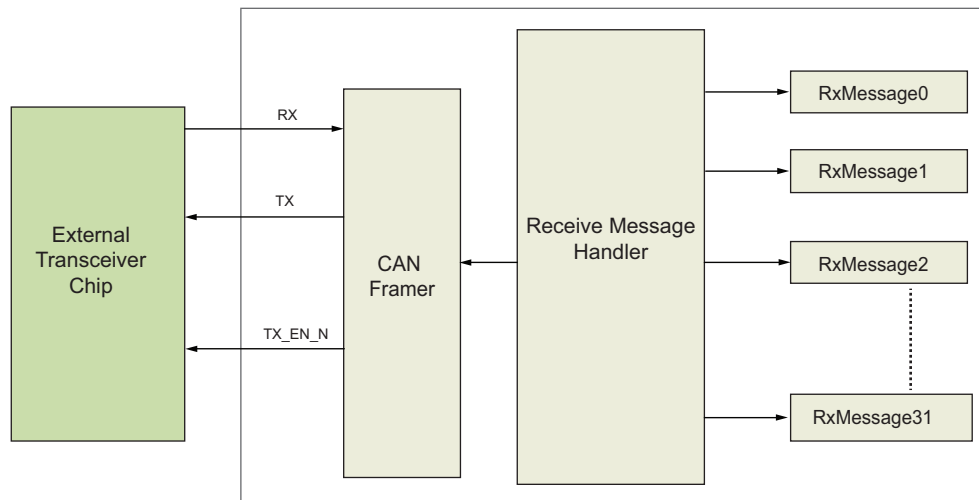
Single-shot transmission (SST) mode is used in systems where the retransmission of a CAN message due to an arbitration loss or a bus error must be prevented. An SST request is set by asserting TxReq and TxAbort at the same time. Upon a successful message transmission, both flags are cleared.

If an arbitration loss or if a bus error happens during the transmission, the TxReq and TxAbort flags are cleared when the message is removed or when the message wins arbitration. At the same time, the sst_failure interrupt is asserted.

12.2.3 Receive Procedures

The CAN controller provides 32 individual receive message buffers. Each one has its own message filter mask. Automatic reply to RTR messages is supported. If a message is accepted in a receive buffer, its MsgAv flag is set. The message remains valid as long as MsgAv flag is set. The host CPU has to reset the MsgAv flag to enable receipt of a new message. The following figure shows the overall block diagram of the receive message buffers.

Figure 168 • Receive Message Buffers



TxMessage1

12.2.3.1 Received Message Processing

After a new message is received, the receive message handler searches all receive buffers, starting from the receive message0 until it finds a valid buffer. A valid buffer is indicated by:

- Receive buffer is enabled (indicated by RxBufferEbl = 1)
- Acceptance filter of receive buffer matches incoming message

If the receive message handler finds a valid buffer that is empty, then the message is stored and the `MsgAv` flag of this buffer is set to 1. If the `RxIntEbl` flag is set, then the `rx_msg` flag of the interrupt controller is asserted.

If the receive buffer already contains a message indicated by `MsgAv = 1` and the link flag is not set, then the `rx_msg_loss` interrupt flag is asserted. Refer to [Receive Buffer Linking](#), page 440.

If an incoming message has its RTR flag set and the RTR reply flag of the matching buffer is set, then the message is not stored but an RTR auto-reply request is issued. Refer to [RTR Auto-Reply](#), page 440 and the `RX_MSG0_CTRL_CMD` register for more details.

12.2.3.2 Acceptance Filter

Each receive buffer has its own acceptance filter that is used to filter incoming messages. An acceptance filter consists of acceptance mask register (AMR) and acceptance code register (ACR) pair. The AMR defines which bits of the incoming CAN message match the corresponding ACR bits.

- The following message fields are covered:
 - ID
 - IDE
 - RTR
 - Data byte 1 and data byte 2

Note: Some CAN HLPs such as Smart Distributed System (SDS) or DeviceNet carry additional protocol related information in the first or first and second data bytes that are used for message acceptance and selection. Having the capability to filter these fields provides a more efficient implementation of the protocol stack running on the Cortex-M3 processor.

The AMR register defines whether the incoming bit is checked against the ACR register. The incoming bit is checked against the respective ACR when the AMR register is 0. The message is not accepted when the incoming bit does not match the respective ACR flag. When the AMR register is 1, the incoming bit is a "don't care".

12.2.3.2.1 RTR Auto-Reply

The CAN controller supports automatic answering of RTR message requests. All 32 receive buffers support this feature. If an RTR message is accepted in a receive buffer where the `RTRreply` flag is set, then this buffer automatically replies to this message with the content of this receive buffer. The `RTRreply` pending flag is set when the RTR message request is received. It is cleared when the message is sent or when the message buffer is disabled. To abort a pending `RTRreply` message, use the `RTRabort` command.

If the RTR auto-reply option is selected, the RTR sent (`RTRS`) flag is asserted when the RTR auto-reply message is successfully sent. It is cleared by writing 1 to it.

An RTR message interrupt is generated if the `RTRS` flag and `RxIntEbl` are set. This interrupt is cleared by clearing the `RTRS` flag.

12.2.3.3 Receive Buffer Linking

Several receive buffers can be linked together to form a receive buffer array which acts almost like a receive FIFO. For a set of receive buffers to be linked together, the following conditions must be met:

- All buffers of the same array must have the same message filter setting (AMR and ACR are identical).
- The last buffer of an array may not have its link flag set.

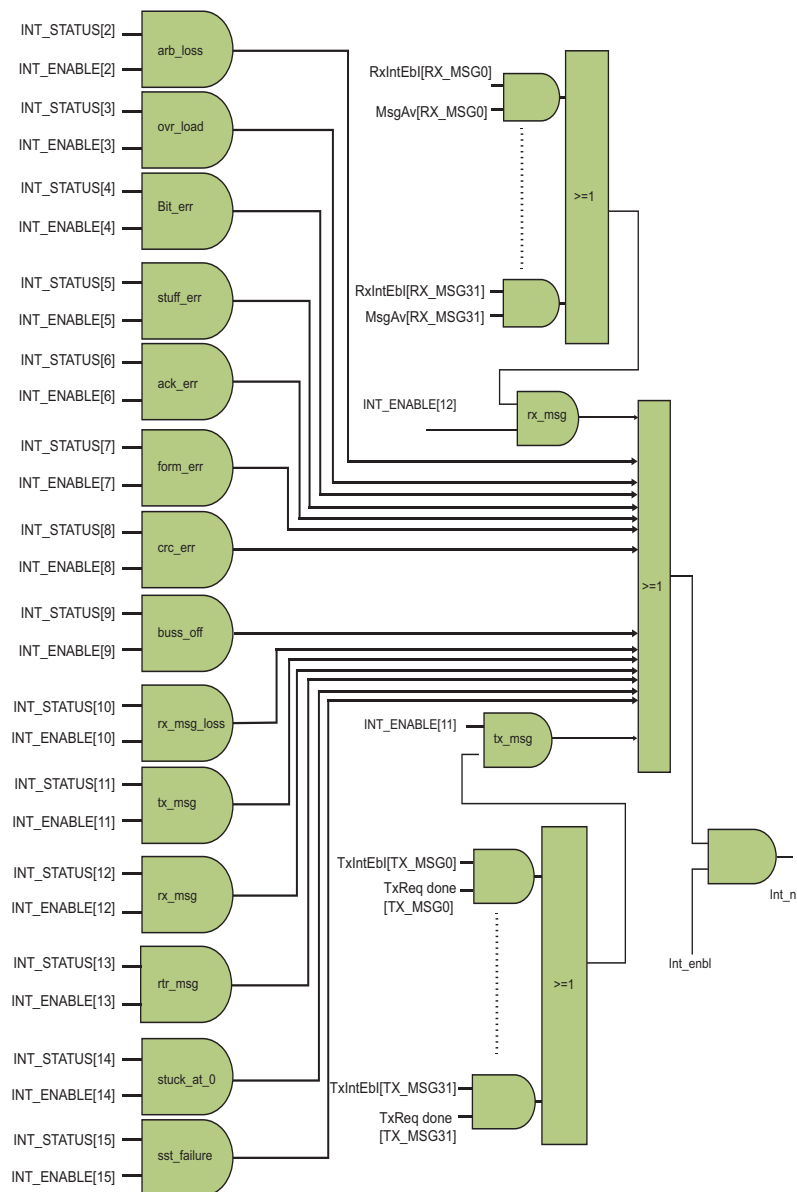
When a receive buffer already contains a message (`MsgAv = 1`) and a new message arrives for this buffer, this message is discarded (`rs_msg_loss` Interrupt). To avoid this situation, several receive buffers can be linked together. When the CAN controller receives a new message, the receive message handler searches for a valid receive buffer. If one is found that is already full (`MsgAv = 1`) and the link flag is set (`LF = 1`); the search for a valid receive buffer continues. If no other buffer is found, the `rx_msg_loss` interrupt is set and the message is discarded.

It is possible to build several message arrays. Each of these arrays must use the same AMR and ACR. The receive buffer locations do not need to be contiguous.

12.2.4 Interrupt Generation

The interrupt `int_n` is asserted, if a particular interrupt status bit and the respective enable bit are set. The following figure shows how the system interrupt is generated.

Figure 169 • Interrupt Generation



12.2.5 CAN Test Modes

Using the [Command Register](#), page 455 loopback and the listen-only settings, the CAN controller can perform certain test operations as summarized in the following table.

Table 436 • Test Modes

Loop back	Listen-only	Comment
0	0	Normal operation
0	1	Listen-only mode The CAN controller receives all bus traffic but does not send any information to the bus. This feature is useful for automatic bit-rate detection.
1	1	Internal loop back The CAN controller receives the sending data. No data is sent to the network and no data is received.
1	0	External loop back The CAN controller participates in the regular CAN transmission and reception. Furthermore, a copy of all sent messages is received. This mode works only if at least one additional CAN node is on the network.

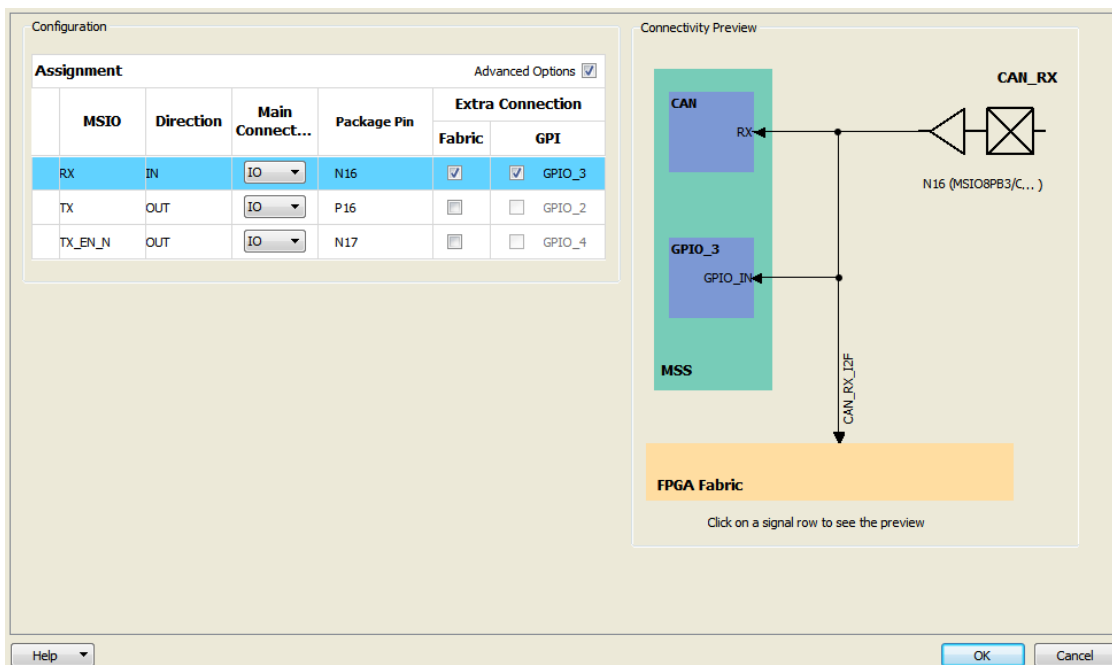
12.3 CAN Controller Configuration

The CAN Controller instance can be accessed from the MSS configurator. Depending on the application, the CAN instance can be enabled or disabled (default) from the MSS. When disabled, the CAN instance is held in reset (lowest power state). When enabled, by default, the CAN ports are configured to connect to the CAN allocated device MSIO CAN ports.

Note: The MSIOs allocated to the CAN instance are shared with other MSS peripherals. These shared I/Os are available to connect to the MSS GPIOs.

Connecting the CAN ports to the allocated CAN MSIOs or fabric or both is done in the CAN Configurator. The Advanced Options in the Configuration window provide the options to enable extra connectivity between the MSIOs and fabric or GPIOs (or both fabric and GPIOs), as shown in the following figure.

Figure 170 • CAN Configurator GUI



12.3.1 Peripheral Signals Assignment Table

The CAN configurator window is divided into two main sections:

- The Configuration window displays all the Assignment Options.
- The Connectivity Preview window shows a graphical view based on the configurations and selections for the highlighted signal row.

There are no hardware configuration options for the CAN peripheral.

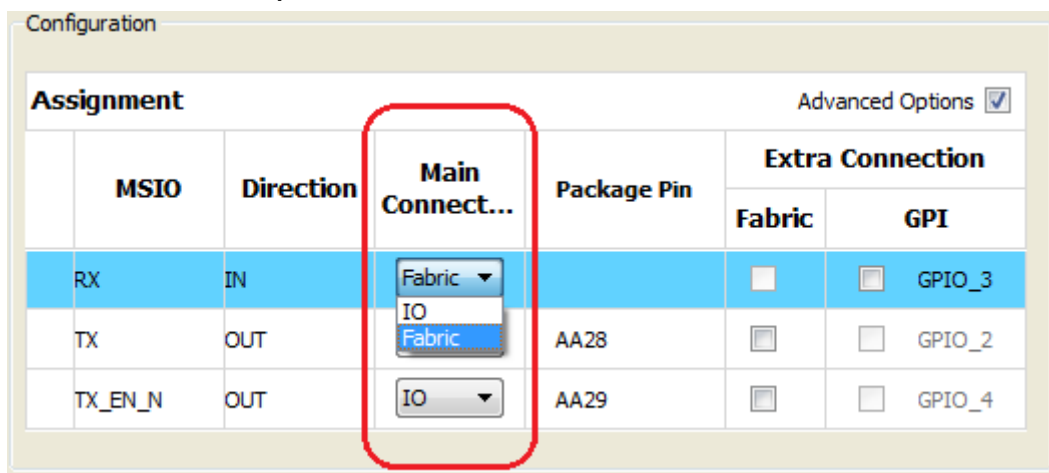
The following table summarizes the valid connections for different configurations that are available in the CAN configurator dialog.

Table 437 • Summary of Different Valid CAN Connections

CAN Port	Direction	Main Connection	Extra Connection	
			Fabric	GPIO
RX	IN	CAN MSIO	Yes	Yes
		Fabric	N/A	N/A
TX	OUT	CAN MSIO	Yes	N/A
		Fabric	N/A	N/A
TX_EN_N	OUT	CAN MSIO	Yes	N/A
		Fabric	N/A	N/A

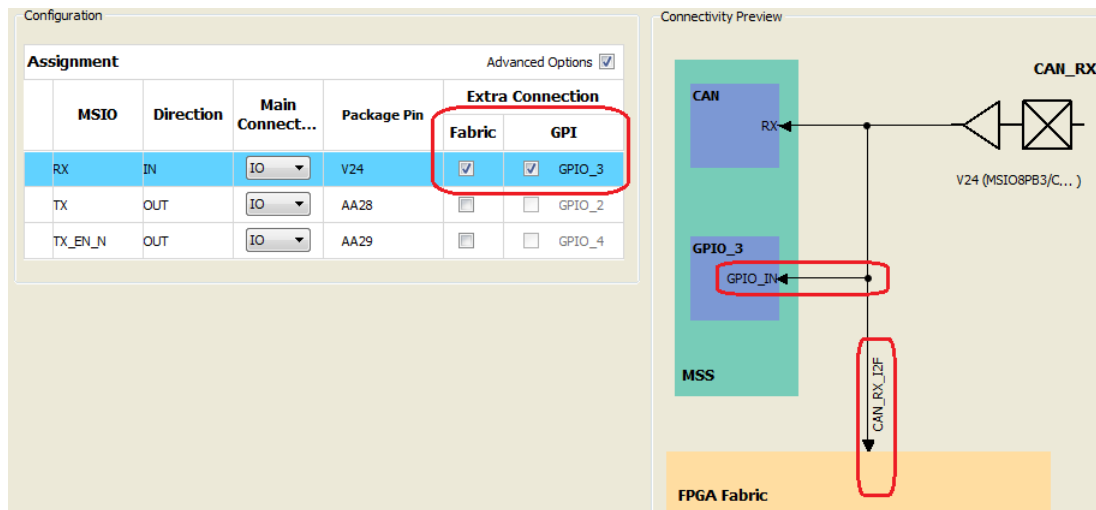
Each port is connected to the allocated CAN MSIO or to the fabric through the Main Connection drop-down menu, as shown in the following figure.

Figure 171 • Main Connection Options - Either MSIO or Fabric



When the main connection is I/O, the signal can also be routed through the fabric by selecting the **Fabric** under Extra Connection. The Extra Connection option is available only when **Advance Options** is selected. The MSIO RX input can also be routed to the GPIO and fabric, as shown in the following figure.

Figure 172 • Extra Connection to Fabric and GPIO Options



Because the MSS peripherals (MMUART, I2C, SPI, CAN, GPIO, USB, Ethernet MAC) share MSIO and FPGA fabric access resources, the configuration of any of these peripherals may result in a resource conflict when an instance of the current peripheral is configured. Peripheral configurators provide clear indicators when such a conflict arises. For more information, refer to the [MSS CAN Configurator User Guide](#).

The functional behavior of the CAN instance must be defined at the application level using the SmartFusion2 MSS CAN firmware driver provided by Microsemi. Refer to the **CAN Firmware Driver User Guide** for more details.

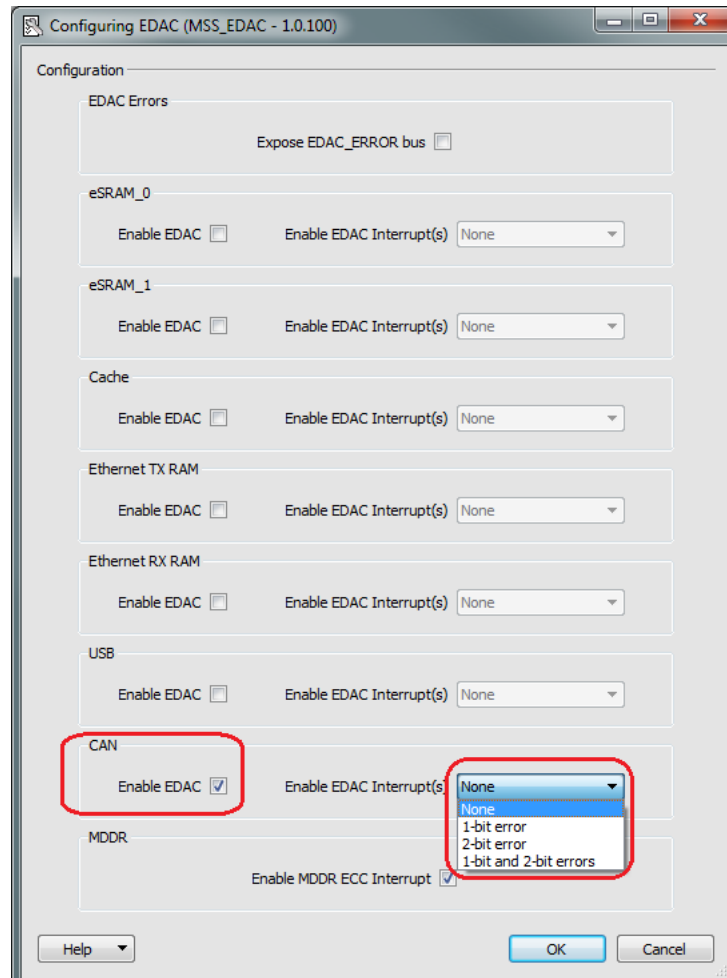
12.3.2 EDAC CAN Configuration

In radiation prone environments, storage elements such as RAMs and FIFOs are susceptible to transient errors caused by heavy ions. Errors can be detected and corrected by employing EDAC. The EDAC controller implemented in SmartFusion2 device supports SECEDED. The CAN controller internal RAMs are one of the RAMs that are protected by EDAC within SmartFusion2 devices.

The values entered in the configurator are exported into the programming files for programming the flash bits that control this functionality. The flash bits are loaded in the system registers at power-up (or when the DEVRST_N external pad is asserted or deasserted).

To configure (enable/disable) EDAC for the CAN controller, the SECDED configurator can be used within the MSS configurator in Libero SoC.

Figure 173 • Enabling EDAC for the CAN from the SECDED Configurator



Expose EDAC_ERROR Bus - Use to expose the EDAC_ERROR bus signal to the FPGA fabric, where it can be used by the design.

Enable EDAC - Use to enable EDAC functionality for the CAN controller.

Enable EDAC Interrupt(s) - Use to enable the EDAC interrupts for the CAN controller. Interrupts for 1-bit error and 2-bit error or both can be enabled, as shown in the preceding figure. Refer to the [MSS EDAC Configuration User Guide](#) for more information on the SECDED configurator options and ports descriptions.

Note: The MSS CAN does not support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for more information.

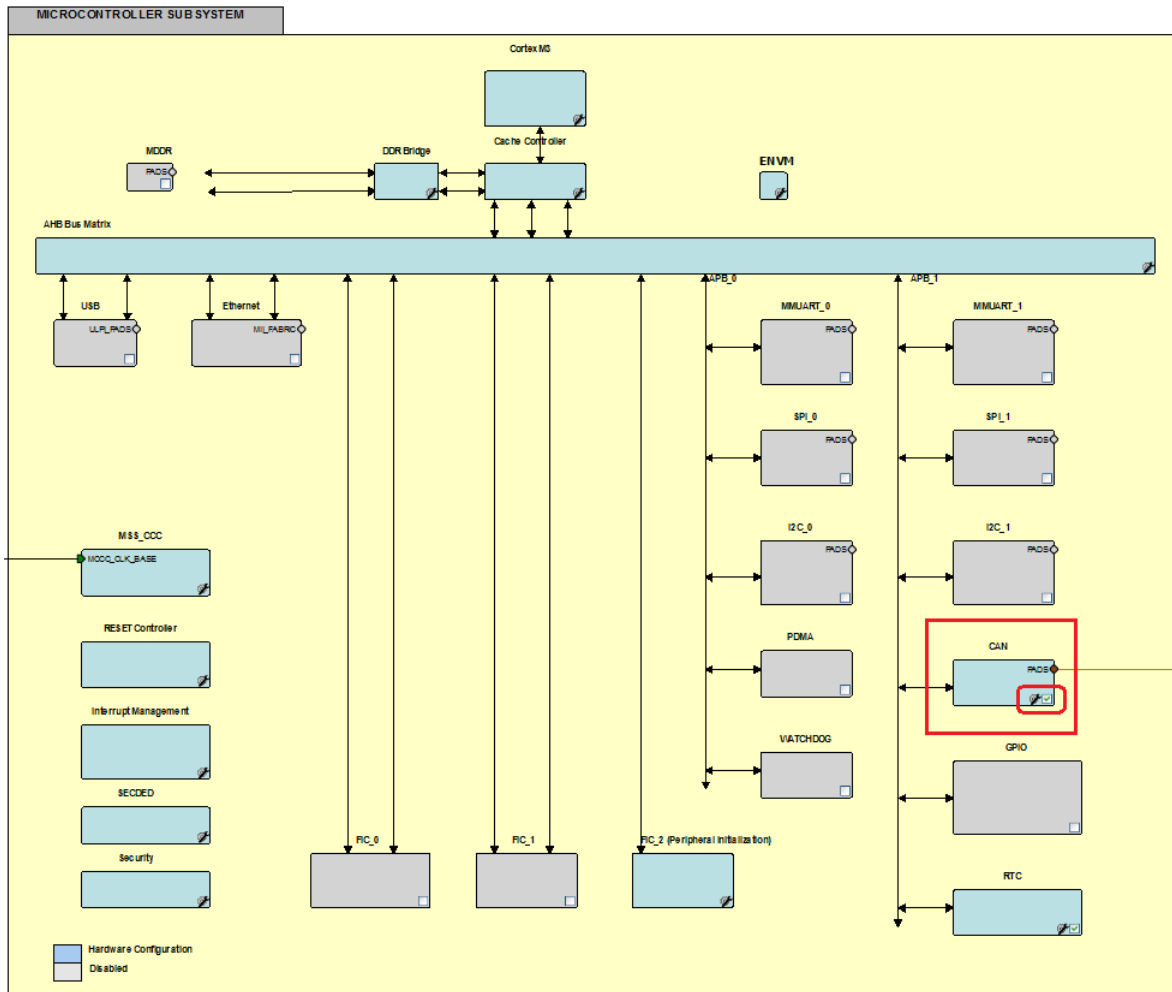
12.4 How to Use the MSS CAN Controller

Follow the below steps to use the CAN controller in an application.

12.4.1 Hardware Design Flow

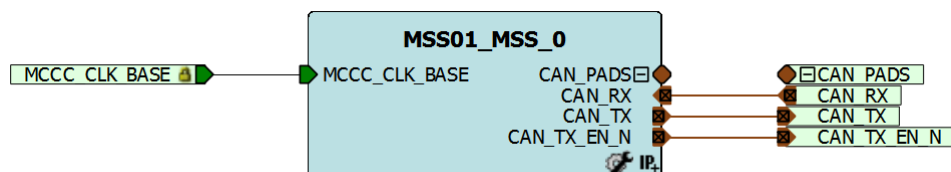
Enable the CAN controller in the MSS configurator for the Libero SoC design project, as shown in the following figure.

Figure 174 • Enabling CAN Controller With MSS Configurator



When the CAN is enabled, CAN_RX, CAN_TX, and CAN_TX_EN_N will be promoted to the top MSS component, as shown in the following figure.

Figure 175 • CAN Signals

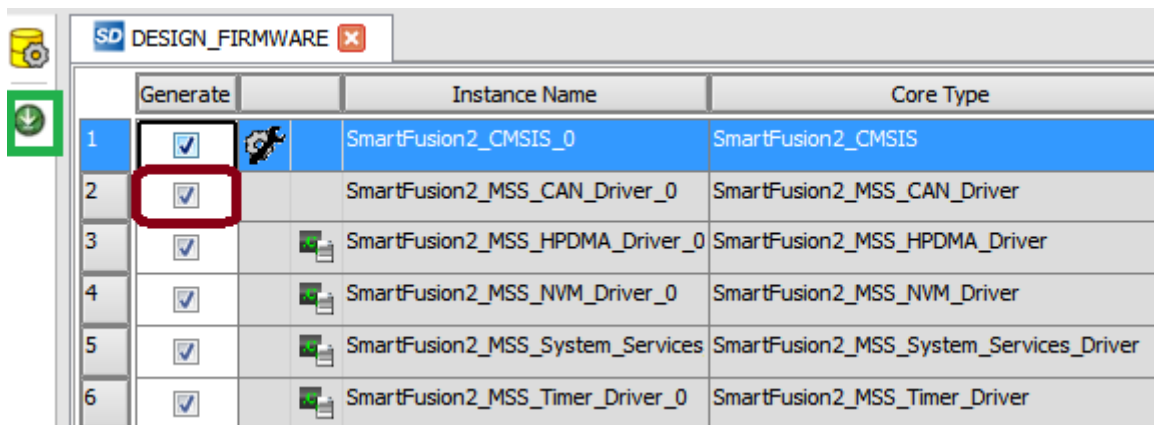


Generate the component by clicking **Generate Component** or by selecting **Generate Component** from the SmartDesign menu. The firmware driver folder and the SoftConsole workspace are generated and included in the project automatically.

For firmware development, double-click **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole Firmware Project. The SoftConsole folder contains the required mss_driver which provides a set of functions for controlling the MSS CAN peripheral.

Note: If the drivers are not generated, make sure that the CAN firmware driver downloaded into the vault from the repositories is available and Generate option is enabled in the **DesignFirmware** window, as shown in the following figure.

Figure 176 • Firmware Driver Enable and Generate

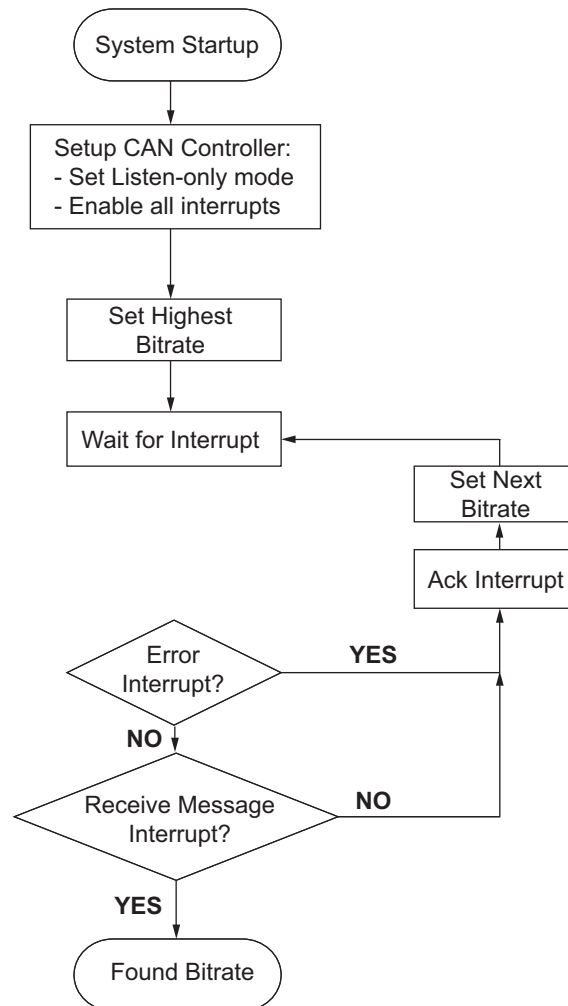


12.5 Use Cases

12.5.1 Use Case 1: Automatic Bit Rate Detection

It is possible to use the CAN in Listen-only test mode. Refer to the [Command Register](#), page 455 and [CAN Test Modes](#), page 442 for more information on how to enable Listen-only mode. In Listen-only mode, the CAN controller receives all bus traffic but does not send any information to the bus. In this non-intrusive bus observation mode, the CAN controller can be used to determine the actual bit rate. During the bit rate detection, the CAN controller listens to the on-going CAN bus communication using a set of given bit rates and eventually detects the actual bit rate. The following flow chart outlines the procedure to detect the bit rate.

Figure 177 • Automatic Bit Rates Detection Flow Chart



12.5.2 Use Case 2: SRAM Test Mode

In addition to the test modes described in the Test Modes section, the CAN controller can be put into SRAM test mode. The CAN controller has a built-in RAM, which is protected by EDAC that is used to store the Receive and Transmit messages.

To support software based memory testing, the CAN controller can be put into SRAM test mode. When this SRAM test mode is active, the CAN controller operation is disabled and transparent access from the host APB interface to all SRAM memory locations is available. SRAM test mode can be enabled or disabled by setting the CAN command bit[3] of the CAN_COMMAND register. Refer to the [Command Register](#), page 455 for more details. SRAM test mode and CAN controller operation are mutually

exclusive. Thus SRAM test mode can only be enabled when CAN controller is stopped and the CAN controller can only be started when the SRAM test mode is stopped. In the SRAM test mode:

- Transparent read and write access to all SRAM memory locations is supported
- All message buffer write protect features are disabled
- Access to receive and transmit message buffer control registers is disabled

In SRAM test mode, the APB interface is used to access different SRAM address directly for test or initialization purposes. At power-up, SRAM is not initialized and any READ action to the memory locations would result in an ECC error if EDAC is enabled. Hence, putting the CAN controller into SRAM test mode enables initialization of the SRAM so ECC errors at power-up do not occur if EDAC is enabled.

The following table provides address mapping between the APB and SRAM addresses.

Table 438 • APB to SRAM Address Mapping

APB Address	SRAM Address	Description
0x020	0x000	TxObject0:Control Bits
0x024	0x001	TxObject0:Identifier Bits
0x028	0x002	TxObject0:Data High Bits
0x02C	0x003	TxObject0:Data Low Bits
0x030-0x03C	0x004-0x007	TxObject1
0x040-0x04C	0x008-0x00B	TxObject2
0x050-0x05C	0x00C-0x00F	TxObject3
0x060-0x06C	0x010-0x013	TxObject4
0x070-0x07C	0x014-0x017	TxObject5
0x080-0x08C	0x018-0x01B	TxObject6
0x090-0x09C	0x01C-0x01F	TxObject7
0x0A0-0x0AC	0x020-0x023	TxObject8
0x0B0-0x0BC	0x024-0x027	TxObject9
0x0C0-0x0CC	0x028-0x02B	TxObject10
0x0D0-0x0DC	0x02C-0x02F	TxObject11
0x0E0-0x0EC	0x030-0x033	TxObject12
0x0F0-0x0FC	0x034-0x037	TxObject13
0x100-0x10C	0x038-0x03B	TxObject14
0x110-0x11C	0x03C-0x03F	TxObject15
0x120-0x12C	0x040-0x043	TxObject16
0x130-0x13C	0x044-0x047	TxObject17
0x140-0x14C	0x048-0x04B	TxObject18
0x150-0x15C	0x04C-0x04F	TxObject19
0x160-0x16C	0x050-0x053	TxObject20
0x170-0x17C	0x054-0x057	TxObject21
0x180-0x18C	0x058-0x05B	TxObject22
0x190-0x19C	0x05C-0x05F	TxObject23
0x1A0-0x1AC	0x060-0x063	TxObject24
0x1B0-0x1BC	0x064-0x067	TxObject25

Table 438 • APB to SRAM Address Mapping (continued)

APB Address	SRAM Address	Description
0x1C0-0x1CC	0x068-0x06B	TxObject26
0x1D0-0x1DC	0x06C-0x06F	TxObject27
0x1E0-0x1EC	0x070-0x073	TxObject28
0x1F0-0x1FC	0x074-0x077	TxObject29
0x200-0x20C	0x078-0x07B	TxObject30
0x210-0x21C	0x07C-0x07F	TxObject31
0x220	0x080	RxObject0:Control Bits
0x224	0x081	RxObject0:Identifier Bits
0x228	0x082	RxObject0:Data High Bits
0x22C	0x083	RxObject0:Data Low Bits
0x230	0x084	RxObject0:AMR - ID
0x234	0x085	RxObject0:ACR - ID
0x238	0x086	RxObject0:AMR - Data
0x23C	0x087	RxObject0:ACR - Data
0x240-0x25C	0x088-0x08F	Receive Message Object 1
0x260-0x27C	0x090-0x097	Receive Message Object 2
0x280-0x29C	0x098-0x09F	Receive Message Object 3
0x2A0-0x2BC	0x0A0-0x0A7	Receive Message Object 4
0x2C0-0x2DC	0x0A8-0x0AF	Receive Message Object 5
0x2E0-0x2FC	0x0B0-0x0B7	Receive Message Object 6
0x300-0x31C	0x0B8-0x0BF	Receive Message Object 7
0x320-0x33C	0x0C0-0x0C7	Receive Message Object 8
0x340-0x35C	0x0C8-0x0CF	Receive Message Object 9
0x360-0x37C	0x0D0-0x0D7	Receive Message Object 10
0x380-0x39C	0x0D8-0x0DF	Receive Message Object 11
0x3A0-0x3BC	0x0E0-0x0E7	Receive Message Object 12
0x3C0-0x3DC	0x0E8-0x0EF	Receive Message Object 13
0x3E0-0x3FC	0x0F0-0x0F7	Receive Message Object 14
0x400-0x41C	0x0F8-0x0FF	Receive Message Object 15
0x420-0x43C	0x100-0x107	Receive Message Object 16
0x440-0x45C	0x108-0x10F	Receive Message Object 17
0x460-0x47C	0x110-0x117	Receive Message Object 18
0x480-0x49C	0x118-0x11F	Receive Message Object 19
0x4A0-0x4BC	0x120-0x127	Receive Message Object 20
0x4C0-0x4DC	0x128-0x12F	Receive Message Object 21
0x4E0-0x4FC	0x130-0x137	Receive Message Object 22
0x500-0x51C	0x138-0x13F	Receive Message Object 23
0x520-0x53C	0x140-0x147	Receive Message Object 24

Table 438 • APB to SRAM Address Mapping (continued)

APB Address	SRAM Address	Description
0x540-0x55C	0x148-0x14F	Receive Message Object 25
0x560-0x57C	0x150-0x157	Receive Message Object 26
0x580-0x59C	0x158-0x15F	Receive Message Object 27
0x5A0-0x5BC	0x160-0x167	Receive Message Object 28
0x5C0-0x5DC	0x168-0x16F	Receive Message Object 29
0x5E0-0x5FC	0x170-0x177	Receive Message Object 30
0x600-0x61C	0x178-0x17F	Receive Message Object 31

12.6 CAN Controller Register Map

12.6.1 SYSREG Control Registers

In addition to the specific CAN registers described in this chapter, the registers found in the CAN SYSREG Control Registers also control the behavior of the CAN peripheral. Refer to the [System Register Block](#), page 670 for a detailed description of each register and bit.

Table 439 • CAN SYSREG Control Registers

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
EDAC_CR	0x38	RW-P	Register	SYSRESET_N	Configures EDAC component of the CAN. To enable or disable the EDAC for the CAN, set the CAN_EDAC_EN bit (6th bit in this register) as follows: 0: EDAC is disabled. 1: EDAC is enabled.
EDAC_IRQ_ENABLE_CR	0x78	RW-P	Register	SYSRESET_N	Configures EDAC interrupts To set 1-bit error or 2-bit error, set the CAN_EDAC_1E_EN and CAN_EDAC_2E_N bits (the 12th and 13th bits in this register) 0: Disables the status signal. 1: Enables the status signal.
CAN_EDAC_CNT	0x108	RO		SYSRESET_N	CAN EDAC count This is a 16-bit counter value in CAN. It is incremented by CAN EDAC 1-bit or 2-bit error. The counter does not roll back and stays at its max value.
CAN_EDAC_ADR	0x11C	RO		SYSRESET_N	CAN EDAC address register CAN memory address on which 1-bit or 2-bit SECEDED error occurs.
EDAC_SR	0x190	SW1C		SYSRESET_N	EDAC status register This status is updated by CAN when a 1-bit or 2-bit SECEDED error has been detected and a single-bit error is corrected for RAM memory.

Table 439 • CAN SYSREG Control Registers (continued)

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
CLR_EDAC_COUNTERS	0x1A4	W1P		SYSRESET_N	Clear EDAC counters This is a pulse generated to clear the 16-bit counter value in CAN corresponding to the count value of EDAC 1-bit or 2-bit errors. This in turn clears the upper 16-bits of CAN_EDAC_CNT register.

At power-up, the CAN_SOFTRESET bit is asserted as 1. This keeps the CAN controller in a reset state. To release the CAN controller from reset, set this bit to 0 as described in [Table 440](#), page 452. If CAN_SOFTRESET is 0, the CAN controller could still be held in reset by other system reset sources. Before specifying the CAN controller configurations, release it from reset.

Table 440 • CAN Controller Soft Reset Bit in the SOFT_RESET_CR Register

Bit Number	Name	R/W	Reset Value	Description
13	CAN_SOFTRESET	R/W	0x1	Controls reset input to CAN Controller 0: Release CAN controller from reset. 1: Keep CAN controller in reset.

12.6.2 CAN Controller Registers

This section describes the register and bit description of various categories of registers in the CAN controller. In addition, system registers which are applicable to CAN are described in this section. This provides programmers the view for firmware development. Microsemi recommends using drivers provided in the tool set for application development.

The CAN base address resides at 0x40015000 and extends to address 0x40015FFF in the Cortex-M3 processor memory map. The registers set in the CAN controller are summarized in [Table 441](#), page 452.

Table 441 • Summary of CAN Controller Registers

Register Name	Address	R/W	Reset Value	Description
CAN_CONFIG	0x018	R/W	0	CAN configuration register. The CAN controller has to be configured prior to its use. The configuration includes effective CAN data rate, CAN data synchronization, and message buffer arbitration. This register has to be configured before the CAN controller is started (Run mode). Run or STOP of the CAN controller is a different action than reset. Refer to the Command Register , page 455 for more details.

Table 441 • Summary of CAN Controller Registers (continued)

Register Name	Address	R/W	Reset Value	Description
CAN_COMMAND	0x014	R/W	0	The CAN controller can be used in different operating modes. By disabling transmitting data, it is possible to use the CAN in Listen-only mode, enabling features such as automatic bit rate detection. Before starting the CAN controller, all the CAN configuration registers have to be set according to the target application.
TX_MSG0_CTRL_CMD	0x020	R/W	0	Transmit Message0 buffer control and command register
TX_MSG0_ID	0x024	R/W	0	Transmit Message0 buffer identifier register
TX_MSG0_DATA_HIGH	0x028	R/W	0	Transmit Message0 buffer data high register
TX_MSG0_DATA_LOW	0x02C	R/W	0	Transmit Message0 buffer data low register
TX_BUF_STATUS	0x0C	R	0	Transmit (TX) message buffer status. This bundles transmit request (TxReq) pending flags from all 32 receive message buffers.
RX_MSG0_CTRL_CMD	0x220	R/W	0	Receive Message0 buffer command and control register
RX_MSG0_ID	0x224	R/W	0	Receive Message0 buffer Identifier register
RX_MSG0_DATA_HIGH	0x228	R/W	0	Receive Message0 buffer data high register
RX_MSG0_DATA_LOW	0x22C	R/W	0	Receive Message0 buffer data low register
RX_MSG0_AMR	0x230	R/W	0	Acceptance mask register (AMR) The AMR register defines whether the incoming bit is checked against the ACR register
RX_MSG0_ACR	0x234	R/W	0	Acceptance code register (ACR)
RX_MSG0_AMR_DATA	0x238	R/W	0	AMR- Data
RX_MSG0_ACR_DATA	0x23C	R/W	0	ACR- Data
RX_BUF_STATUS	0x08	R	0	Receive (RX) message buffer status. This bundles message available (MsgAv) flags from all 32 receive message buffers.
ECR	0x01C	RW	0	Error capture register Can be used to perform additional CAN bus diagnostics
ERROR_STATUS	0x010	R	0	CAN error status indicator register Provides visibility into CAN controller error state, receive error count, and transmit error count. Special flags to report error counter values equal to or in excess of 96 errors are available to indicate heavily disturbed bus situations.
INT_ENABLE	0x004	R/W	0	Interrupt enable register Writing 1 to a particular bit enables the corresponding interrupt source as set in INT_STATUS.

Table 441 • Summary of CAN Controller Registers (continued)

Register Name	Address	R/W	Reset Value	Description
INT_STATUS	0x00	R/W	0x00	Interrupt status register Writing 1 to a particular bit sets the corresponding interrupt source. The associated enable bit in INT_ENABLE must also be set for this interrupt to be generated.

The following sections describe the functionality and the bit description of each of the registers in more details.

12.6.3 Configuration Register

The CAN configuration register CAN_CONFIG is a 32-bit register that is used to configure the functionality of the CAN controller. The CAN controller has to be configured prior to its use. The following table shows the CAN_CONFIG register and summarizes the bit description within the register.

Table 442 • CAN_CONFIG

Bit Number	Name	Description
31	Reserved	Reserved
[30:16]	CFG_BITRATE	Configuration bit rate Prescaler for generating the time quantum which defines the TQ: 0: One time quantum equals 1 clock cycle 1: One time quantum equals 2 clock cycles 32767: One time quantum equals 32768 clock cycles
15	Reserved	Reserved
14	ECR_MODE	Error-capture mode 0: Free running. The ECR register shows the current bit position within the CAN frame. 1: Capture mode. The ECR register shows the bit position and type of the last captured CAN error.
13	SWAP_ENDIAN	The byte position of the CAN receive and transmit data fields can be modified to match the endian setting of the processor or the used CAN protocol. 0: CAN data byte position is not swapped (big endian). 1: CAN data byte position is swapped (little endian).
12	CFG_ARBITER	Transmit buffer arbiter 0: Round robin arbitration. 1: Fixed priority arbitration.
[11:8]	CFG_TSEG1	Time segment 1. Time segment 1 includes the propagation time Length of the first time segment: $tseg1 = CFG_TSEG1 + 1$ CFG_TSEG1 = 0 and CFG_TSEG1 = 1 are not allowed.
[7:5]	CFG_TSEG2	Time segment 2 Length of the second time segment: $tseg2 = CFG_TSEG2 + 1$ CFG_TSEG2 = 0 is not allowed. CFG_TSEG2 = 1 is only allowed in Direct-sampling mode.
4	AUTO_RESTART	The CAN can be set to restart either 'by hand' or automatically after a bus-off. 0: After bus-off, the CAN must be restarted 'by hand'. This is the recommended setting. 1: After bus-off, the CAN restarts automatically after 128 groups of 11 recessive bits.

Table 442 • CAN_CONFIG (continued)

Bit Number	Name	Description
[3:2]	CFG_SJW	Synchronization Jump Width 1 sjw ≤ tseg1 and sjw ≤ tseg2
1	SAMPLING_MODE	CAN bus bit sampling 0: One sampling point is used in the receiver path. 1: Three sampling points with majority decision are used.
0	EDGE_MODE	CAN bus synchronization logic 0: Edge from 'R' to 'D' is used for synchronization. 1: Both edges are used.

12.6.4 Command Register

The CAN controller can be set to operate in different modes by setting the command register CAN_COMMAND, as shown in the following table.

Table 443 • CAN_COMMAND

Bit Number	Name	Reset Value	Function
[31:16]	Revision_Control		This field contains the version of the CAN core in the following format. This is a read-only field. [major version].[minor version].[revision number] [31:28]: Major version [27:24]: Minor version [23:16]: Revision number
[15:4]	Reserved	0	Reserved
[3:0]	CAN_COMMAND[3]	0	SRAM Test mode 0: Normal operation 1: Enable SRAM Test mode.
	CAN_COMMAND[2]	0	Loopback-test mode 0: Normal operation 1: Loopback mode is enabled[1]: Listen-only mode.
	CAN_COMMAND[1]	0	Listen-only mode 0: Active 1: CAN listen only. The output is held at 'R' level. The CAN controller is only listening.
	CAN_COMMAND[0]	0	Run/Stop mode 0: Sets the CAN controller into Stop mode. Returns 0 when stopped. 1: Sets the CAN controller into Run mode. Returns 1 when running.

12.6.5 Transmit Message Control and Command Register

Each transmit buffer can be configured through a set of registers. Those registers are broken down into a Control/Command register, Identifier register, Data high register, and Data low register. In the Command/Control register, some bits are setting a control flag and others are setting a command flag. The following tables provide a detailed description of transmit message0 buffer registers bits.

Note: The rest of the transmit messages (message1 to message31) follow the same registers definition as message0.

Table 444 • TX_MSG0_CTRL_CMD

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0	Reserved
23	WPN	1	Write protect not 0: Bit[21:16] remain unchanged 1: The write protect is not set and bit[21:16] are modified, by default. The read back value of this bit is undefined Using the WPN flag enables simple retransmission of the same message by only having to set the TxReq and TxAbort flags without taking care of the special flags
22	Reserved	0	Reserved
21	RTR	0	RTR; control flag bit 0: Standard message 1: RTR message
20	IDE	0	Extended identifier bit; Control flag bit 0: This is a standard format message 1: This is an extended format message
[19:16]	DLC	0	Data length code; Control flag bit Invalid values are transmitted as they are, but the number of data bytes is limited to eight 0: Message has 0 data bytes 1: Message has 1 data byte ... 8: Message has 8 data bytes 9-15: Message has 8 data bytes
[15:4]	Reserved	0	Reserved
3	WPN	1	Write protect not 0: Bit[2] remains unchanged. 1: The write protect is not set and Bit[2] is modified, default.
2	TxIntEbl	0	Tx interrupt enable; Control flag bit 0: Interrupt is disabled. 1: Interrupt enabled, successful message transmission sets the TX_MSG flag in the interrupt controller.
1	TxAbort	0	Transmit abort request; Command flag bit 0: Idle 1: Requests removal of a pending message. The message is removed the next time an arbitration loss happens. The flag is cleared when the message is removed or when the message wins arbitration. The TxReq flag is cleared at the same time.

Table 444 • TX_MSG0_CTRL_CMD (continued)

Bit Number	Name	Reset Value	Description
0	TxReq	0	Transmit request; Command flag bit Write: 0: Idle. No message transmit request. 1: Message transmit request The Tx message buffer must not be changed while TxReq is 1. Read: 0: TxReq completed 1: TxReq pending

Table 445 • TX_MSG0_ID

Bit Number	Name	Reset Value	Description
[31:3]	ID[28:0]	0	Transmit message0 buffer identifier (29-bit wide)
[2:0]	Reserved	0	Reserved

Table 446 • TX_MSG0_DATA_HIGH

Bit Number	Name	Reset Value	Description
[31:0]	TX_MSG0_DATA_HIGH	[31:24]: CAN data byte 1 [23:16]: CAN data byte 2 [15:8]: CAN data byte 3 [7:0]: CAN data byte 4	The byte mapping can be set using the CAN swap_endian configuration bit. swap_endian = 0, default [31:24]: CAN data byte 1 [23:16]: CAN data byte 2 [15:8]: CAN data byte 3 [7:0]: CAN data byte 4 swap_endian = 1 [31:24]: CAN data byte 4 [23:16]: CAN data byte 3 [15:8]: CAN data byte 2 [7:0]: CAN data byte 1

Table 447 • TX_MSG0_DATA_LOW

Bit Number	Name	Reset Value	Description
[31:0]	TX_MSG0_DATA_LOW	[31:24]: CAN data byte 5 [23:16]: CAN data byte 6 [15:8]: CAN data byte 7 [7:0]: CAN data byte 8	The byte mapping can be set using the CAN swap_endian configuration bit swap_endian = 0, default [31:24]: CAN data byte 5 [23:16]: CAN data byte 6 [15:8]: CAN data byte 7 [7:0]: CAN data byte 8 swap_endian = 1 [31:24]: CAN data byte 8 [23:16]: CAN data byte 7 [15:8]: CAN data byte 6 [7:0]: CAN data byte 5

The rest of the transmit message buffers (TX_MSG1 to TX_MSG31) registers and registers bits have the same descriptions as the TX_MSG0 registers shown above.

The following table lists the address offsets for the TX_MSG1 to TX_MSG31 registers.

Table 448 • Transmit Message1 to Transmit Message31 Registers Description

Register Name	Address Offset	R/W	Reset Value	Description
TX_MSG1 Buffer	0x030-0x03C	R/W	0	Transmit Message1 buffer registers
TX_MSG2 Buffer	0x040-0x04C	R/W	0	Transmit Message2 buffer registers
TX_MSG3 Buffer	0x050-0x05C	R/W	0	Transmit Message3 buffer registers
TX_MSG4 Buffer	0x060-0x06C	R/W	0	Transmit Message4 buffer registers
TX_MSG5 Buffer	0x070-0x07C	R/W	0	Transmit Message5 buffer registers
TX_MSG6 Buffer	0x080-0x08C	R/W	0	Transmit Message6 buffer registers
TX_MSG7 Buffer	0x090-0x09C	R/W	0	Transmit Message7 buffer registers
TX_MSG8 Buffer	0x0A0-0x0AC	R/W	0	Transmit Message8 buffer registers
TX_MSG9 Buffer	0x0B0-0x0BC	R/W	0	Transmit Message9 buffer registers
TX_MSG10 Buffer	0x0C0-0x0CC	R/W	0	Transmit Message10 buffer registers
TX_MSG11 Buffer	0x0D0-0x0DC	R/W	0	Transmit Message11 buffer registers
TX_MSG12 Buffer	0x0E0-0x0EC	R/W	0	Transmit Message12 buffer registers
TX_MSG13 Buffer	0x0F0-0x0FC	R/W	0	Transmit Message13 buffer registers
TX_MSG14 Buffer	0x100-0x10C	R/W	0	Transmit Message14 buffer registers
TX_MSG15 Buffer	0x110-0x11C	R/W	0	Transmit Message15 buffer registers
TX_MSG16 Buffer	0x120-0x12C	R/W	0	Transmit Message16 buffer registers
TX_MSG17 Buffer	0x130-0x13C	R/W	0	Transmit Message17 buffer registers
TX_MSG18 Buffer	0x140-0x14C	R/W	0	Transmit Message18 buffer registers
TX_MSG19 Buffer	0x150-0x15C	R/W	0	Transmit Message19 buffer registers
TX_MSG20 Buffer	0x160-0x16C	R/W	0	Transmit Message20 buffer registers
TX_MSG21 Buffer	0x170-0x17C	R/W	0	Transmit Message21 buffer registers
TX_MSG22 Buffer	0x180-0x18C	R/W	0	Transmit Message22 buffer registers
TX_MSG23 Buffer	0x190-0x19C	R/W	0	Transmit Message23 buffer registers
TX_MSG24 Buffer	0x1A0-0x1AC	R/W	0	Transmit Message24 buffer registers
TX_MSG25 Buffer	0x1B0-0x1BC	R/W	0	Transmit Message25 buffer registers
TX_MSG26 Buffer	0x1C0-0x1CC	R/W	0	Transmit Message26 buffer registers
TX_MSG27 Buffer	0x1D0-0x1DC	R/W	0	Transmit Message27 buffer registers
TX_MSG28 Buffer	0x1E0-0x1EC	R/W	0	Transmit Message28 buffer registers
TX_MSG29 Buffer	0x1F0-0x1FC	R/W	0	Transmit Message29 buffer registers
TX_MSG30 Buffer	0x200-0x20C	R/W	0	Transmit Message30 buffer registers
TX_MSG31 Buffer	0x210-0x21C	R/W	0	Transmit Message31 buffer registers

Refer to TX_MSG0 buffer for description from [Table 444](#), page 456 through [Table 447](#), page 457.

12.6.6 Transmit Buffer Status Register

The following table lists the transmit buffer status indicator register TX_BUF_STATUS bits descriptions of the register. For transmit, the status indicates if a message is ready to be sent out. This register consolidate the status of all transmit message buffers.

Table 449 • TX_BUF_STATUS

Bit Number	Name	Reset Value	Description
31	TxMessage31	0	Message available in TxMessage buffer 31
30	TxMessage30	0	Message available in TxMessage buffer 30
29	TxMessage29	0	Message available in TxMessage buffer 29
28	TxMessage28	0	Message available in TxMessage buffer 28
27	TxMessage27	0	Message available in TxMessage buffer 27
26	TxMessage26	0	Message available in TxMessage buffer 26
25	TxMessage25	0	Message available in TxMessage buffer 25
24	TxMessage24	0	Message available in TxMessage buffer 24
23	TxMessage23	0	Message available in TxMessage buffer 23
22	TxMessage22	0	Message available in TxMessage buffer 22
21	TxMessage21	0	Message available in TxMessage buffer 21
20	TxMessage20	0	Message available in TxMessage buffer 20
19	TxMessage19	0	Message available in TxMessage buffer 19
18	TxMessage18	0	Message available in TxMessage buffer 18
17	TxMessage17	0	Message available in TxMessage buffer 17
16	TxMessage16	0	Message available in TxMessage buffer 16
15	TxMessage15	0	Message available in TxMessage buffer 15
14	TxMessage14	0	Message available in TxMessage buffer 14
13	TxMessage13	0	Message available in TxMessage buffer 13
12	TxMessage12	0	Message available in TxMessage buffer 12
11	TxMessage11	0	Message available in TxMessage buffer 11
10	TxMessage10	0	Message available in TxMessage buffer 10
9	TxMessage9	0	Message available in TxMessage buffer 9
8	TxMessage8	0	Message available in TxMessage buffer 8
7	TxMessage7	0	Message available in TxMessage buffer 7
6	TxMessage6	0	Message available in TxMessage buffer 6
5	TxMessage5	0	Message available in TxMessage buffer 5
4	TxMessage4	0	Message available in TxMessage buffer 4
3	TxMessage3	0	Message available in TxMessage buffer 3
2	TxMessage2	0	Message available in TxMessage buffer 2
1	TxMessage1	0	Message available in TxMessage buffer 1
0	TxMessage0	0	Message available in TxMessage buffer 0

12.6.7 Receive Message Control and Command Register

Each receive buffer can be configured through a set of registers. Those registers are broken down into a Command/Control register, Identifier register, Data high register, and Data low register, Acceptance mask register (AMR), Acceptance code register (ACR), AMR data, and ACR data. The following tables provide a detailed description of receive message0 buffer registers bits.

Table 450 • RX_MSG0_CTRL_CMD

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0	Reserved
23	WPNH	1	Write protect not high 0: Bit[21:16] remain unchanged 1: The write protect is not set and bit[21:16] are modified, default The read back value of this bit is undefined
22	Reserved	0	Reserved
21	RTR	0	RTR bit; Control bit 0: This is a regular message 1: This is an RTR message.
20	IDE	0	Extended identifier bit; Control bit 0: This is a standard format message. 1: This is an extended format message.
[19:16]	DLC	0	Data length code; Control bits 0: Message has 0 data byte. 1: Message has 1 data byte. ... 8: Message has 8 data bytes. 9-15: Message has 8 data bytes.
[15:8]	Reserved	0	Reserved
7	WPNL	1	Write protect not low 0: Bits[6:3] remain unchanged 1: This write protect is not set and bits[6:3] are modified, default. This bit is always zero for read back
6	LF	0	Link flag; Control bit 0: This buffer is not linked to the next buffer 1: This buffer is linked with the next buffer
5	RxIntEbl	0	Receive interrupt enable; Control bit 0: Interrupt generation is disabled 1: Interrupt generation is enabled
4	RTRreply	0	Automatic message reply upon receipt of an RTR message; Control bit 0: Automatic RTR message handling disabled 1: Automatic RTR message handling enabled
3	TxBufferEbl	0	Transaction buffer enable; Control bit 0: Buffer is disabled 1: Buffer is enabled

Table 450 • RX_MSG0_CTRL_CMD (continued)

Bit Number	Name	Reset Value	Description
2	RTRabort	0	RTR abort request; Command bit 0: Idle 1: Requests removal of a pending RTR message reply. The flag is cleared when the message was removed or when the message won arbitration. The TxReq flag is cleared at the same time
1	RTRP	0	RTRreply pending; Command bit 0: No RTR reply request pending 1: RTR reply request pending
0	MsgAv/RTRS	0	Message available/RTR sent; Command bit If RTRreply flag is set, this bit shows if an RTR auto-reply message has been sent, otherwise it indicates if the buffer contains a valid message. Read 0: Idle 1: New message available (RTRreply = 0), RTR auto-reply message sent (RTRreply = 1) Write 0: Idle 1: Acknowledges receipt of new message or transmission of RTR auto-reply message. Before acknowledging receipt of a new message, the message content must be copied into system memory. Acknowledging a message clears the MsgAv flag.

Table 451 • RX_MSG0_ID

Bit Number	Name	Reset Value	Description
[31:3]	ID[28:0]		RxMessage0 buffer identifier (29-bit wide)
[2:0]	Reserved	0	N/A

Table 452 • RX_MSG0_DATA_HIGH

Bit Number	Name	Reset Value	Description
[31:0]	RX_MSG0_DATA_HIGH	[31:24]: CAN data byte 1 [23:16]: CAN data byte 2 [15:8]: CAN data byte 3 [7:0]: CAN data byte 4	The byte mapping can be set using the CAN swap_endian configuration bit. swap_endian = 0, default: [31:24]: CAN data byte 1 [23:16]: CAN data byte 2 [15:8]: CAN data byte 3 [7:0]: CAN data byte 4 swap_endian = 1 [31:24]: CAN data byte 4 [23:16]: CAN data byte 3 [15:8]: CAN data byte 2 [7:0]: CAN data byte 1

Table 453 • RX_MSG0_DATA_LOW

Bit Number	Name	Reset Value	Description
[31:0]	RX_MSG0_DATA_LOW	[31:24]: CAN data byte 5 [23:16]: CAN data byte 6 [15:8]: CAN data byte 7 [7:0]: CAN data byte 8	The byte mapping can be set using the CAN swap_endian configuration bit. swap_endian = 0, default: [31:24]: CAN data byte 5 [23:16]: CAN data byte 6 [15:8]: CAN data byte 7 [7:0]: CAN data byte 8 swap_endian = 1 [31:24]: CAN data byte 8 [23:16]: CAN data byte 7 [15:8]: CAN data byte 6 [7:0]: CAN data byte 5

Table 454 • RX_MSG0_AMR

Bit Number	Name	Reset Value	Description
[31:0]	RX_MSG0_AMR		Receive Message0 buffer AMR bits [31:3]: Identifier [2]: IDE [1]: RTR [0]: Reserved AMR: 0: The incoming bit is checked against the respective ACR. The message is not accepted when the incoming bit does not match with the respective ACR flag. 1: The incoming bit is a "don't care"

Table 455 • RX_MSG0_ACR

Bit Number	Name	Reset Value	Description
[31:0]	RX_MSG0_ACR		Receive Message0 buffer ACR bits [31:3]: Identifier [2]: IDE [1]: RTR [0]: N/A

Table 456 • RX_MSG0_AMR_DATA

Bit Number	Name	Reset Value	Description
[31:0]	RX_MSG0_AMR_DATA		Receive Message0 buffer AMR Data bits [15:8]: CAN data byte 1 [7:0]: CAN data byte 2

Table 457 • RX_MSG0_ACR_DATA

Bit Number	Name	Reset Value	Description
[31:0]	RX_MSG0_ACR_DATA		Receive Message0 buffer ACR Data bits [15:8]: CAN data byte 1 [7:0]: CAN data byte 2

The rest of the receive message buffers (RX_MSG1 to RX_MSG31) and register bits have the same descriptions as the RX_MSG0 registers shown above. The following table lists the address offset for the RX_MSG1 to RX_MSG31 registers.

Table 458 • Receive Message1 to Receive Message 31 and ECR Registers Description

Register Name	Address Offset	R/W	Reset Value	Description
RX_MSG1 Buffer	0x240-0x25C	R/W	0	Receive Message1 buffer registers
RX_MSG2 Buffer	0x260-0x27C	R/W	0	Receive Message2 buffer registers
RX_MSG3 Buffer	0x280-0x29C	R/W	0	Receive Message3 buffer registers
RX_MSG4 Buffer	0x2A0-0x2BC	R/W	0	Receive Message4 buffer registers
RX_MSG5 Buffer	0x2C0-0x2DC	R/W	0	Receive Message5 buffer registers
RX_MSG6 Buffer	0x2E0-0x2FC	R/W	0	Receive Message6 buffer registers
RX_MSG7 Buffer	0x300-0x31C	R/W	0	Receive Message7 buffer registers
RX_MSG8 Buffer	0x320-0x33C	R/W	0	Receive Message8 buffer registers
RX_MSG9 Buffer	0x340-0x35C	R/W	0	Receive Message9 buffer registers
RX_MSG10 Buffer	0x360-0x37C	R/W	0	Receive Message10 buffer registers
RX_MSG11 Buffer	0x380-0x39C	R/W	0	Receive Message11 buffer registers
RX_MSG12 Buffer	0x3A0-0x3BC	R/W	0	Receive Message12 buffer registers
RX_MSG13 Buffer	0x3C0-0x3DC	R/W	0	Receive Message13 buffer registers
RX_MSG14 Buffer	0x3E0-0x3FC	R/W	0	Receive Message14 buffer registers
RX_MSG15 Buffer	0x400-0x41C	R/W	0	Receive Message15 buffer registers
RX_MSG16 Buffer	0x420-0x43C	R/W	0	Receive Message16 buffer registers
RX_MSG17 Buffer	0x440-0x45C	R/W	0	Receive Message17 buffer registers
RX_MSG18 Buffer	0x460-0x47C	R/W	0	Receive Message18 buffer registers
RX_MSG19 Buffer	0x480-0x49C	R/W	0	Receive Message19 buffer registers
RX_MSG20 Buffer	0x4A0-0x4BC	R/W	0	Receive Message20 buffer registers
RX_MSG21 Buffer	0x4C0-0x4DC	R/W	0	Receive Message21 buffer registers
RX_MSG22 Buffer	0x4E0-0x4FC	R/W	0	Receive Message22 buffer registers
RX_MSG23 Buffer	0x500-0x51C	R/W	0	Receive Message23 buffer registers
RX_MSG24 Buffer	0x520-0x53C	R/W	0	Receive Message24 buffer registers
RX_MSG25 Buffer	0x540-0x55C	R/W	0	Receive Message25 buffer registers
RX_MSG26 Buffer	0x560-0x57C	R/W	0	Receive Message26 buffer registers
RX_MSG27 Buffer	0x580-0x59C	R/W	0	Receive Message27 buffer registers
RX_MSG28 Buffer	0x5A0-0x5BC	R/W	0	Receive Message28 buffer registers
RX_MSG29 Buffer	0x5C0-0x5DC	R/W	0	Receive Message29 buffer registers
RX_MSG30 Buffer	0x5E0-0x5FC	R/W	0	Receive Message30 buffer registers
RX_MSG31 Buffer	0x600-0x61C	R/W	0	Receive Message31 buffer registers

Refer to RX_MSG0 buffer for description from [Table 450](#), page 460 through [Table 457](#), page 462.

12.6.8 Receive Buffer Status Register

The following table lists the receive buffer status indicator register RX_BUF_STATUS and the bits descriptions of the register. For receive buffer, the status indicates if a message arrived. This register consolidates the status of all receive message buffers.

Table 459 • RX_BUF_STATUS

Bit Number	Name	Reset Value	Description
31	RxMessage31	0	Message available in RxMessage buffer 31
30	RxMessage30	0	Message available in RxMessage buffer 30
29	RxMessage29	0	Message available in RxMessage buffer 29
28	RxMessage28	0	Message available in RxMessage buffer 28
27	RxMessage27	0	Message available in RxMessage buffer 27
26	RxMessage26	0	Message available in RxMessage buffer 26
25	RxMessage25	0	Message available in RxMessage buffer 25
24	RxMessage24	0	Message available in RxMessage buffer 24
23	RxMessage23	0	Message available in RxMessage buffer 23
22	RxMessage22	0	Message available in RxMessage buffer 22
21	RxMessage21	0	Message available in RxMessage buffer 21
20	RxMessage20	0	Message available in RxMessage buffer 20
19	RxMessage19	0	Message available in RxMessage buffer 19
18	RxMessage18	0	Message available in RxMessage buffer 18
17	RxMessage17	0	Message available in RxMessage buffer 17
16	RxMessage16	0	Message available in RxMessage buffer 16
15	RxMessage15	0	Message available in RxMessage buffer 15
14	RxMessage14	0	Message available in RxMessage buffer 14
13	RxMessage13	0	Message available in RxMessage buffer 13
12	RxMessage12	0	Message available in RxMessage buffer 12
11	RxMessage11	0	Message available in RxMessage buffer 11
10	RxMessage10	0	Message available in RxMessage buffer 10
9	RxMessage9	0	Message available in RxMessage buffer 9
8	RxMessage8	0	Message available in RxMessage buffer 8
7	RxMessage7	0	Message available in RxMessage buffer 7
6	RxMessage6	0	Message available in RxMessage buffer 6
5	RxMessage5	0	Message available in RxMessage buffer 5
4	RxMessage4	0	Message available in RxMessage buffer 4
3	RxMessage3	0	Message available in RxMessage buffer 3
2	RxMessage2	0	Message available in RxMessage buffer 2
1	RxMessage1	0	Message available in RxMessage buffer 1
0	RxMessage0	0	Message available in RxMessage buffer 0

12.6.9 Error Capture Register

The CAN controller contains a dedicated error capture register (ECR) that can be used to perform additional CAN bus diagnostics. The ECR supports two different modes.

- Free-running mode: In Free-running mode, the ECR displays the field and bit position within the current CAN frame
- Error-capture mode: In Error-capture mode, the ECR displays the bit position and type of the last captured CAN error.

In Error-capture mode, the ECR samples the field and bit position, when a CAN error is detected. In order to sample such an event, the ECR needs to be set by performing a write access to it. When the CAN is set to sample the event, the ECR only captures one error event. For successive error captures, the ECR needs to be set again. The following table provides the bits descriptions in the ECR register.

Table 460 • ECR

Bit Number	Name	Reset Value	Description
[31:17]	Reserved	0	Reserved
[16:12]	Field	0	This specifies the field of the ECR 0x00: Stopped 0x01: Synchronize 0x05: Interframe 0x06: Bus idle 0x07: Start of frame 0x08: Arbitration 0x09: Control 0x0A: Data 0x0B: CRC 0x0C: ACK 0x0D: End of frame 0x10: Error flag 0x11: Error echo 0x12: Error delimiter 0x18: Overload flag 0x19: Overload echo 0x1A: Overload delimiter Others: N/A
[11:6]	Bit_number	0	Bit number inside of field
5	Rx_mode	0	When asserted, the CAN controller is the receiver.
4	Tx_mode	0	When asserted, the CAN controller is the transmitter.
[3:1]	Error_type	0	Specifies different error types 0: Arbitration loss 1: Bit error 2: Bit stuffing error 3: Acknowledge error 4: Form error 5: CRC error Others: N/A
0	Status	0	Status of the ECR register 0: The ECR register captured an error or is in free running mode. 1: The ECR register set to sample the event.

12.6.10 Error Status Register

Status indicators are provided to report the CAN controller error state, receive error count, and transmit error count. Special flags to report error counter values equal to or in excess of 96 errors are available to indicate heavily disturbed bus situations.

The following table provides ERROR_STATUS register bit descriptions.

Table 461 • ERROR_STATUS

Bit Number	Name	Reset Value	Description
[31:20]	Reserved	0	Reserved
19	rxgte96	0	The receive error counter is greater than or equal to 96 dec.
18	txgte96	0	The transmit error counter is greater than or equal to 96 dec.
[17:16]	error_state[1:0]	0	The error state of the CAN mode: 00: error active (normal operation) 01: error passive 1x: bus off
[15:8]	rx_err_cnt[7:0]	0	The receive error counter as defined in CAN 2.0 specification. When in bus-off state, this counter is used to count 128 groups of 11 receive bits.
[7:0]	tx_err_cnt[7:0]	0	The transmit error counter as defined in CAN 2.0 specification. When it is greater than 255 dec, it is fixed at 255 dec.

12.6.11 Interrupt Registers

The interrupt controller contains an interrupt enable (INT_ENABLE) and an interrupt status (INT_STATUS) registers.

12.6.11.1 Interrupt Enable Register

The interrupt enable register controls which particular bits from the interrupt status register are used to assert the interrupt output int_n. Refer to [Interrupt Generation](#), page 441 for more details. The bits in the INT_ENABLE register control, which bits in the INT_STATUS register are used to enable the final output, int_n, interrupt. The Interrupt int_n is asserted if a particular interrupt status bit and the respective enable bit are set. The following table provides the INT_ENABLE register bit descriptions.

Table 462 • INT_ENABLE

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	Reserved
15	sst_failure_enbl	0	Single shot transmission failure interrupt enable.
14	stuck_at_0_enbl	0	Stuck at dominant error interrupt enable.
13	rtr_msg_enbl	0	RTR auto-reply message sent interrupt enable.
12	rx_msg_enbl	0	Receive message available interrupt enable.
11	tx_msg_enbl	0	Message transmitted interrupt enable.
10	rx_msg_loss_enbl	0	Received message lost interrupt enable.
9	bus_off_enbl	0	Bus off interrupt enable.
8	crc_err_enbl	0	CRC error interrupt enable.
7	form_err_enbl	0	Format error interrupt enable.
6	ack_err_enbl	0	Acknowledge error interrupt enable.

Table 462 • INT_ENABLE (continued)

Bit Number	Name	Reset Value	Description
5	stuff_err_enbl	0	Bit stuffing error interrupt enable.
4	bit_err_enbl	0	Bit error interrupt enable.
3	ovr_load_enbl	0	Overload message detected interrupt enable.
2	arb_loss_enbl	0	Arbitration loss interrupt enable.
1	Reserved	0	Reserved
0	Int_enbl	0	Global interrupt enable flag. 0: All interrupts are disabled 1: Enabled interrupt sources are available

12.6.11.2 Interrupt Status Register

The interrupt status register stores internal interrupt events. Once a bit is set, it remains set until it is cleared by writing 1 to it. The interrupt enable register has no effect on the interrupt status register. The following table provides INT_STATUS register bit descriptions. A pending interrupt indicates that its respective flag is set to 1. In order for an interrupt to be acknowledged, set its flag to 1.

Table 463 • INT_STATUS

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	Reserved
15	SST_FAILURE	0	Single-shot transmission failure 0: Normal operation 1: A buffer set for single shot transmission experienced an arbitration loss or a bus error during transmission.
14	STUCK_AT_0	0	Stuck at dominant error 0: Normal operation 1: Indicates if receive (RX) input remains stuck at 0 (dominant level) for more than 11 consecutive bit times.
13	RTR_MSG	0	RTR auto-reply message sent 0: Normal operation 1: Indicates that a RTR auto-reply message was sent.
12	RX_MSG	0	Receive message available 0: Normal operation 1: Indicates a new message was successfully received and stored in a receive buffer which has its RxIntEbl flag asserted.
11	TX_MSG	0	Message transmitted 0: Normal operation 1: Indicates a message was successfully sent from a transmit buffer which has its TxIntEbl flag asserted.
10	RX_MSG_LOSS	0	Received message lost 0: Normal operation 1: Indicates a newly received message couldn't be stored because the target message buffer was full (for example, its MsgAv flag was set).
9	BUS_OFF	0	Bus Off 0: Normal operation 1: Indicates that the CAN controller entered the bus-off error state.

Table 463 • INT_STATUS (continued)

Bit Number	Name	Reset Value	Description
8	CRC_ERR	0	CRC error 0: Normal operation 1: Indicates that a CAN CRC error is detected.
7	FORM_ERR	0	Format error 0: Normal operation 1: Indicates that a CAN format error is detected.
6	ACK_ERR	0	Acknowledge error 0: Normal operation 1: Indicates that a CAN message acknowledgment error is detected.
5	STUFF_ERR	0	Bit stuffing error 0: Normal operation 1: Indicates that a CAN bit stuffing error is detected.
4	BIT_ERR	0	Bit error 0: Normal operation 1: Indicates that a CAN bit error is detected.
3	OVR_LOAD	0	Overload message detected 0: Normal operation 1: Indicates that a CAN overload message is detected.
2	ARB_LOSS	0	Arbitration loss 0: Normal operation 1: The message arbitration was lost while sending a message. The message transmission is retried once the CAN bus is idle again.
[1:0]	Reserved	0	Reserved

13 MMUART Peripherals

Multi-mode universal asynchronous/synchronous receiver/transmitter (MMUART) performs serial-to-parallel conversion on data originating from modems or other serial devices, and performs parallel-to-serial conversion on data from the Cortex-M3 processor or fabric master to these devices. SmartFusion2 SoC FPGAs contain two identical MMUART peripherals in the microcontroller subsystem (MSS MMUART_0 and MSS MMUART_1), that provide software compatibility with the popular 16550 UART device.

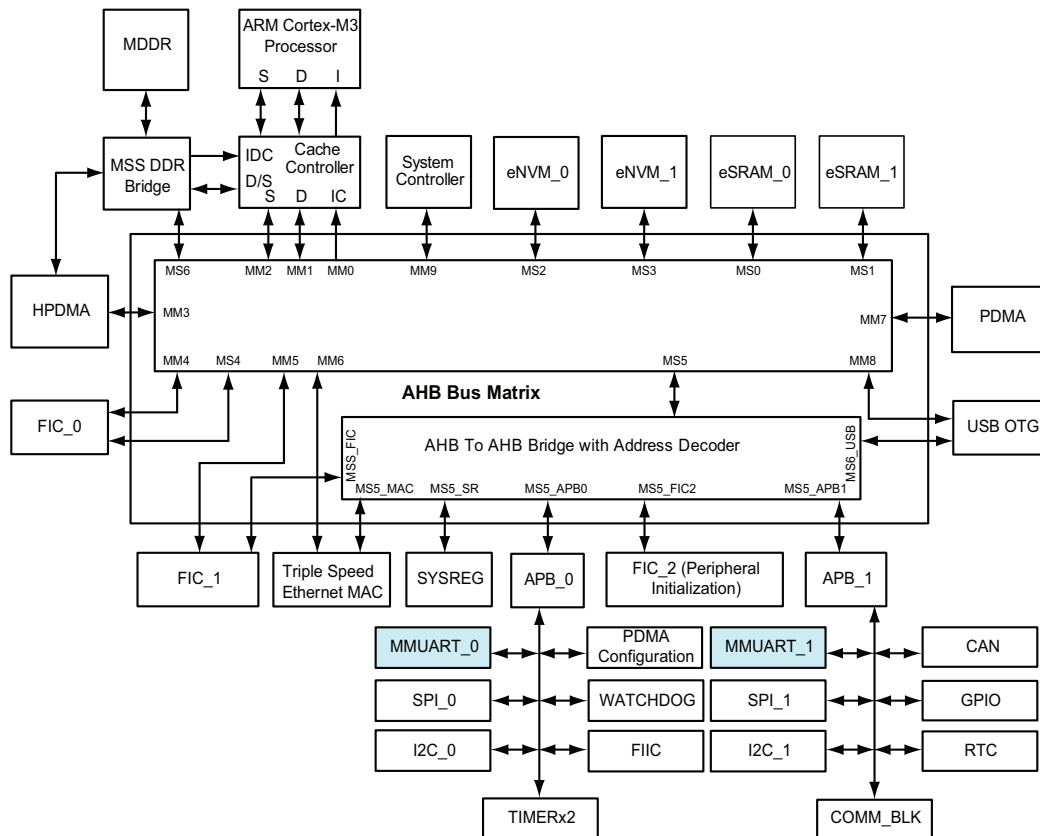
13.1 Features

SmartFusion2 MMUART peripherals support the following features:

- Asynchronous and synchronous operations
- Full programmable serial interface characteristics
 - Data width is programmable to 5, 6, 7, or 8 bits
 - Even, odd, or no-parity bit generation/detection
 - 1, 1½, and 2 stop bit generation
- 9-bit address flag capability used for multi-drop addressing topologies
- Separate transmit (Tx) and receive (Rx) FIFOs to reduce processor interrupt service loading
- Single-wire half-duplex mode in which Tx pad can be used for bi-directional data transfer
- Local interconnect network (LIN) header detection and auto baud rate calculation
- Communication with ISO 7816 smart cards
- Fractional baud rate capability
- Return to Zero Inverted (RZI) mod/demod blocks that allow infrared data association (IrDA) and serial infrared (SIR) communications
- The most significant bit (MSB) or the least significant bit (LSB) as the first bit while sending or receiving data

The following figure shows the MMUART peripherals within the MSS. The MMUART peripherals are interfaced to the AHB bus matrix through APB interfaces (APB_0 and APB_1).

Figure 178 • MSS Showing MMUART Peripherals



13.2 Functional Description

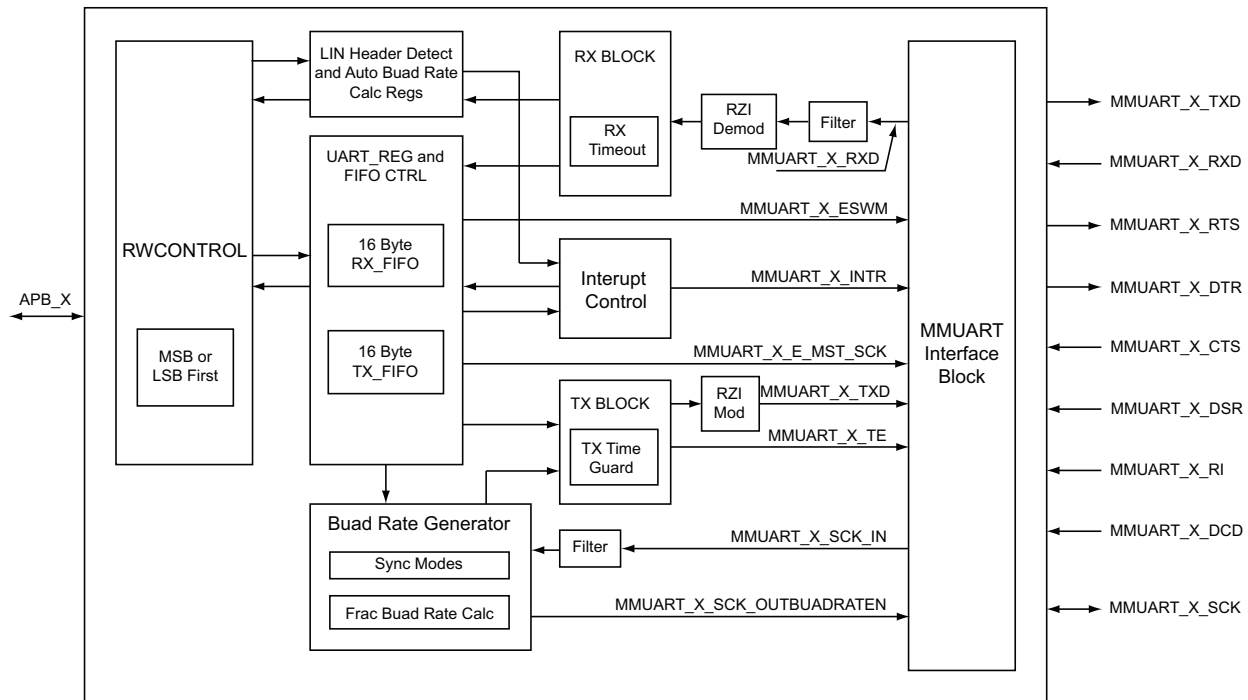
This section provides the detailed description of the MMUART peripherals.

13.2.1 Architecture Overview

The functional block diagram of MMUART is shown in the following figure. The main components of MMUART include Transmit and Receive FIFOs (TX_FIFO and RX_FIFO), baud rate generator, input filters, LIN Header Detection and Auto Baud Rate Calculation block, RZI modulator and demodulator, and interrupt controller.

While transmitting data, the parallel data is written to TX_FIFO of the MMUART to transmit in serial form. While receiving data to RX_FIFO, the MMUART transforms the serial input data into parallel form to facilitate reading by the Cortex-M3 processor.

The baud rate generator contains free running counters and utilizes the asynchronous and synchronous baud rate generation circuits. The input filters in MMUART suppress the noise and spikes of incoming clock signals and serial input data based on the filter length. The RZI modulation/demodulation blocks are intended to allow for IrDA serial infrared (SIR) communications. The operational details of these sub blocks are explained in detail in [Details of Operation](#), page 474.

Figure 179 • MMUART Block Diagram

13.2.2 Port List

The following table lists MMUART I/O signals. X is used as a place holder for 0 or 1 in register and signal descriptions, indicating MMUART_0 or MMUART_1.

Table 464 • MMUART I/O Signal Descriptions

Name	Type	Polarity	Description
MMUART_X_CTS	Input	Low	Clear to send. This signal is used in the modem interface. The active Low signal is an input that indicates when the attached device (modem) is ready to accept data. MMUART passes the information to the Cortex-M3 processor through the modem status register (MSR). The register indicates that the CTSn signal is changed since the register is read last time. This signal can either go to the fabric or to the I/O pad.
MMUART_X_DSR	Input	Low	Data set ready. This signal is used in modem interface. The active Low signal is an input that indicates when the attached device (modem) is ready to set up a link with MMUART. MMUART passes the information to the Cortex-M3 processor through the MSR . This register also indicates that the DSRn signal is changed since the register was read last time. This signal can either go to the fabric or to the I/O pad.
MMUART_X_DCD	Input	Low	Data carrier detect. This signal is used in the modem interface. The active Low signal is an input that indicates when the attached device (modem) has detected a carrier. MMUART passes this information to the Cortex-M3 processor through the MSR . This register also indicates that the DCDn signal is changed since the register is read last time. This signal can either go to the fabric or to the I/O pad.

Table 464 • MMUART I/O Signal Descriptions (continued)

Name	Type	Polarity	Description
MMUART_X_RI	Input	Low	Ring indicator. This signal is used in the modem interface. The active Low signal is an input showing when the attached device (modem) senses a ring signal on the telephone line. MMUART passes this information to the Cortex-M3 processor through the MSR . This register also gives an indication when the RI trailing edge is sensed. This signal can either go to the fabric or to the I/O pad.
MMUART_X_RTS	Output	Low	Request to send. This signal is used in the modem interface. The active Low output signal is used to inform the attached device (modem) that MMUART is ready to send data. It is programmed by the Cortex-M3 processor through the MCR . This signal can either go to the fabric or to the I/O pad.
MMUART_X_DTR	Output	Low	Data terminal ready. This signal is used in the modem interface. The active Low output signal informs the attached device (modem) that MMUART is ready to establish a communications link. It is programmed by the Cortex-M3 processor through the MCR . This signal can either go to the fabric or to the I/O pad.
MMUART_X_RXD	Input		Serial input data This is the data transmitted into MMUART. It is synchronized with the APB clock (master clock) input pin. This signal can either go to the fabric or to the I/O pad.
MMUART_X_TXD	Output		Serial output data. This is the data transmitted from MMUART. It is synchronized with the BAUDOUT output pin. This signal can either go to the fabric or to the I/O pad.
MMUART_X_SCK_IN	Input		Serial input synchronous clock. The MMUART_X_SCK can be configured as an input synchronous clock from master when the MMUART_X acts as a slave.
MMUART_X_SCK_OUT / BAUDOUTN	Output	Low	In Synchronous mode, it is the serial output clock, SCK_OUT. MMUART_X_SCK can be configured as output synchronous clock to a slave when MMUART_X acts as a master. In Asynchronous mode, it is the baud rate clock derived from APB_X_CLK and BAUDOUTN.
MMUART_X_ESWM	Output	High	Single-wire, half-duplex enable. If this signal is active High, then the data I/O is configured for half-duplex operation over a single-wire, using the Tx pad (out signal). If Low, then the data I/O has separate Tx (out) and Rx (in) pins. Single-wire mode enable (ESWM) signal goes to the FPGA fabric, when it can be used on any pad.
MMUART_X_TE	Output	High	Transmitter output enable. This active High signal is used as a bi-directional enable for single-wire half-duplex operation. An active High signal transmits out, and a Low signal is received. TE usage implies that the MMUART is configured in single-wire mode with the single-wire mode (SWM) bit.

Table 464 • MMUART I/O Signal Descriptions (continued)

Name	Type	Polarity	Description
MMUART_X_E_MST_SCK	Output	High	Enable master clock. This active High signal is used as a bi-directional enable for the SCK_IN and SCK_OUT signals. If these signals are taken from a single bi-directional pad, E_MST_SCK active High designates Master mode and forces the bi-directional pad as an output, otherwise (in case of active Low) the pad is an input for SCK_IN.
MMUART_X_OUT1	Output	Low	Output 1. This active Low output is a user-defined signal. It is programmed by the processor via the MCR and is set to the opposite value. It can be used as a bi-directional pad enable for bi-directional topology use models.
MMUART_X_OUT2	Output	Low	Output 2. This active Low output signal is a user-defined signal. It is programmed by the processor via the MCR and is set to the opposite value.

13.2.3 Initialization

This sub-section describes the MMUART initialization sequence, reset, clock requirements, and interrupts. The MMUART can be initialized by configuring the MMUART control registers and SOFT_RESET_CR system registry.

13.2.3.1 MMUART Initialization Sequence

1. Release the MMUART from reset by using SOFT_RESET_CR system registry ([Table 465](#), page 474)
2. Disable the MMUART interrupts by using Interrupt Enable Register ([IER Table 475](#), page 494)
3. Clear the Transmit and receive FIFO of MMUART by using FIFO Control Register ([FCR Table 470](#), page 491)
4. Clear the loopback and remote loopback modes by using Modem Control Register ([MCR Table 481](#), page 497)
5. Configure MMUART to send/receive MSB or LSB as a first bit by using Multi-Mode Control Register 1 ([MM1 Table 486](#), page 501)
6. Set default transmit and receive ready by using ENABLE_TXRDY_RXRDY bit of FIFO Control Register ([FCR Table 470](#), page 491)
7. Disable 9-bit address flag mode and single wire mode by using Multi-Mode Control Register 2 ([MM2 Table 487](#), page 501)
8. Disable transmit time guard and fractional baud rate by using Multi-Mode Control Register 0 ([MM0 Table 485](#), page 500).
9. Set default Receive timeout by using Multi-Mode Control Register 0 (MM0) and Receiver Timeout Register ([RTO Table 490](#), page 503)
10. Set transmit time guard by using Transmitter Time Guard Register ([TTG Table 489](#), page 503),
11. Set input filter length to suppress spikes by using Glitch Filter Register ([GFR Table 488](#), page 502)
12. Configure the baud rate of MMUART by using Baud Rate Registers ([DLR](#), [DMR](#), and [DFR](#)).
13. Set the word length, stop bits and parity of MMUART by using Line Control Register ([LCR Table 480](#), page 496)
14. Disable LIN header detection and automatic baud rate calculation, RZI modulation/demodulation and smart card modes by using ELIN([MM0](#)), EIRD([MM1](#)), and EERR([MM2](#)) bits.

13.2.3.2 MMUART Reset

MMUART resets to zero on power-up and is held in reset until it is enabled. An option is provided under software control to reset the MMUART by writing to bit 7 or bit 8 of the `SOFT_RESET_CR`, located in the `SYSREG` block. At power-up, the reset signals are asserted 1. This keeps MMUART peripherals in a reset state. MMUART peripheral becomes active when the bit is set to 0, as mentioned in the following table.

Table 465 • Soft Reset Bit Definition for the MMUART_x

Bit Number	Name	R/W	Reset Value	Description
8	MMUART1_SOFTRESET	R/W	0x1	Controls reset input to MMUART_1 0: Release MMUART_1 from reset 1: Keep MMUART_1 in reset (reset value)
7	MMUART0_SOFTRESET	R/W	0x1	Controls reset input to MMUART_0 0: Release MMUART_0 from reset 1: Keep MMUART_0 in reset (reset value)

13.2.3.3 Clock Requirements

The MMUART_0 and MMUART_1 peripherals are clocked by `APB_0_CLK` on APB bus 0 and `APB_1_CLK` on APB bus 1. These clocks are derived from the main MSS clock `M3_CLK`. Each APB clock can be programmed individually as `M3_CLK` divided by 1, 2, 4, or 8. Refer to the [UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide](#) for more information on clocks.

The baud rate generator block receives the input from APB clock and divides by the value of the Baud Rate Registers (`DLR`, `DMR`, and `DFR`). The result is then divided further by 16 to produce the integer baud rate. The resultant signal is the `BAUDOUT` signal. The MMUART also has a fractional baud rate generation capability. These features are described in detail in [Baud Rate Generation](#), page 474.

13.2.3.4 Interrupts

There is one interrupt signal from each MMUART peripheral. The `MMUART_0_INT` signal is generated by MMUART_0 and is mapped to `INTISR[10]` in the Cortex-M3 processor nested vectored interrupt controller (NVIC). The `MMUART_1_INT` signal is generated by MMUART_1 and is mapped to `INTISR[11]` in the Cortex-M3 processor NVIC. Both interrupt enable bits within NVIC-`INTISR[10]` and `INTISR[11]` correspond to bit locations 10 and 11. MMUART interrupts can be enabled by setting the appropriate bits in the IER register while the divisor latch access bit of `LCR` (Table 480, page 496) is 0. Ensure the clearing of the appropriate bit in the IER interrupt service routine to prevent a re-assertion of the interrupt. It should be noted that there is currently no priority scheme to interrupt within the MMUART peripheral.

13.2.4 Details of Operation

This sub-section provides functional description of MMURAT internal components and different modes of MMUART operation.

13.2.4.1 Baud Rate Generation

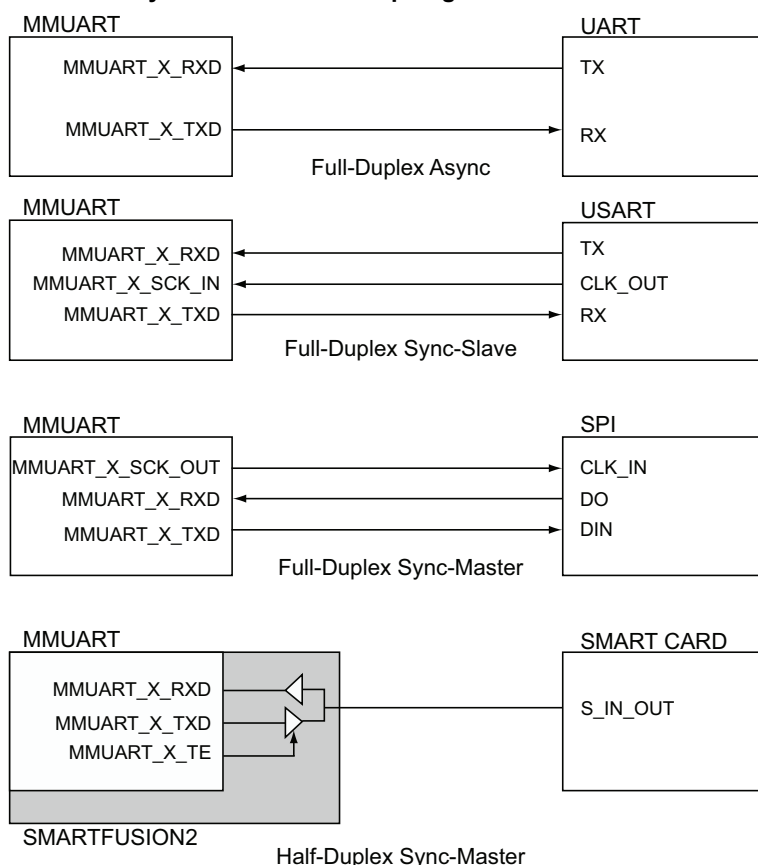
The baud rate generator contains free-running counters that generate the internal clocks. The design utilizes an asynchronous baud rate generation circuit and also allows for synchronous Slave and Master modes.

The three major baud rate generation modes are listed below:

1. **Fractional Baud Rate Generation—Asynchronous and Synchronous—Master Mode:** The baud rate generation follows asynchronous operation, but with an averaging function which yields more precise, overall baud rate values in fractions of system clock cycles.
2. **Clock-In Adaptation—Synchronous—Slave Mode:** In this mode, the MMUART_X_SCK_IN (serial input synchronous clock) determines the baud rate. MMUART may receive and transmit data based on this clock (full-duplex).
3. **Clock-Out Generation—Synchronous—Master Mode:** In this mode, MMUART_X_SCK_OUT (serial output synchronous clock) determines baud rate based on the baud rate registers. MMUART may receive and transmit data based on this clock (full duplex).

Software may drive half-duplex communication based on the application, and a single line may be used for MMUART_X_TXD and MMUART_X_RXD transmissions through a bi-directional pad. While transmitting data, the reception is inhibited and vice-versa. This can be done by enabling the single-wire half-duplex enable (MMUART_X_ESWM). The following figure illustrates different MMUART system topologies.

Figure 180 • Synchronous and Asynchronous Mode Topologies



13.2.4.1.1 Fractional Baud Rate Generation—Asynchronous Mode

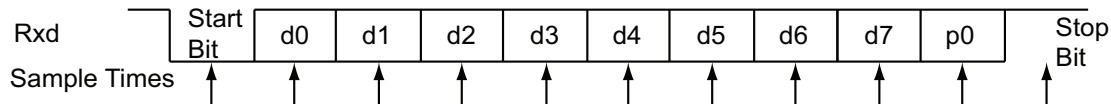
The following figure depicts accumulative baud rate difference error in the Asynchronous mode. It also describes how fractional baud rate generation reduces this error by adding wait times, in the essence of forming a new average baud rate value that is between two given integer baud rate values.

Specifically, fractional baud rate generation modulates the integer baud rate with the integer baud rate + $1/16^{\text{th}}$ of a Tbit time (bit transmit time) to yield a time averaged fractional value. The fractional resolution is $1/64^{\text{th}}$ resolution. A 6-bit phase accumulator evenly distributes the fractional value over 64 Tbit times.

Figure 181 • Sample Time Correction with Fractional Baud Rate

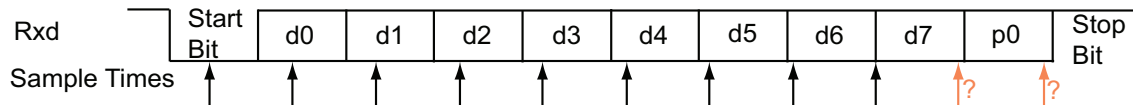
Received Baud Rate Matches Transmitted Baud Rate

*No Sample Error



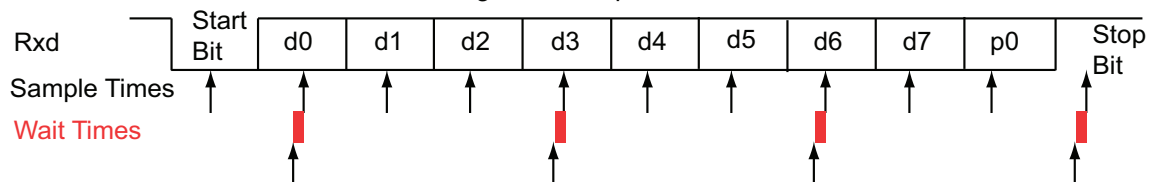
Received Baud Rate Faster Than Transmitted Baud Rate

*Accumulative Error results in Error on the Last Bits



Received Baud Rate Matches Transmitted Baud Rate through injection of single APB CLK cycle waits in the baud generation

*Accumulative Error is Reduced resulting in no Sample Error



The following equation is used to calculate the fractional baud rate:

$$\text{FractionalBaudRate} = \frac{\text{ABP_X_CLK}}{16(\text{DMR} + \text{DLR} + (\text{DFR}/64))}$$

where,

APB_X_CLK: Input reference clock (APB_0_CLK for MMUART_0 and APB_1_CLK for MMUART_1)

DMR: Divisor latch for MSB

DLR: Divisor latch for LSB

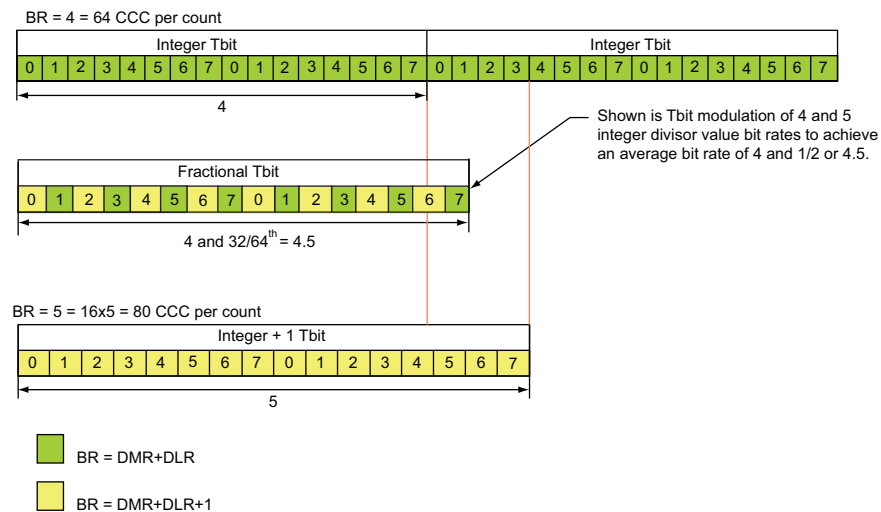
DFR: Fractional divisor register

If a baud rate value of 1 and $1/64^{\text{th}}$ (1.015625) is required, set the "DLR" to 1, the "DMR" to 0, and "DFR" to 1. Following examples illustrate the use of fractional baud rate.

The following figure describes the timing for a fractional baud rate value of 4.5. Exact averaging is accomplished in one Tbit (bit time) cycle.

Figure 182 • Example with Fractional Baud Rate of 4.5

Bit Rate Modulation with Tbit Times



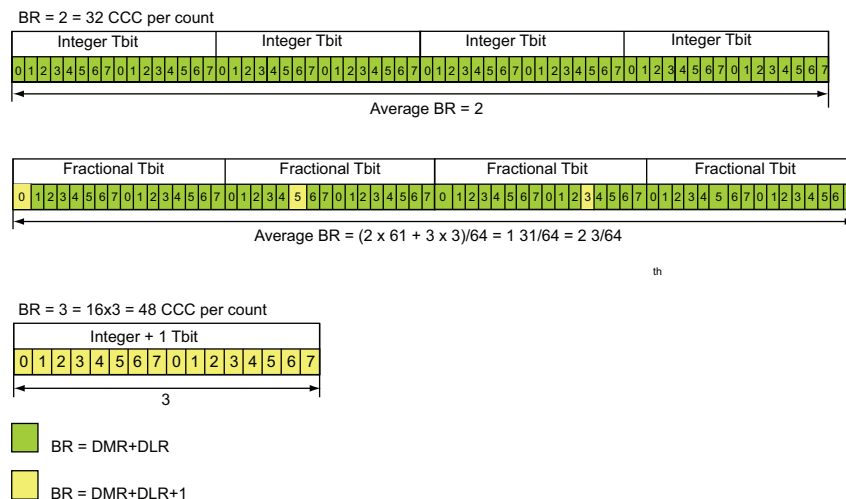
Notes:

- Lowest Common Denominator Value of 32 requires two Tbit times to complete the time modulation fractional baud rate. Lowest Common Denominator Value of 64 requires four Tbit times to complete the time averaged fractional baud rate. Thus, depending on the fraction chosen, the Tbit time may or may not be constant for 2 or 4 Tbit times max, but the time averaged Tbit time will be correct.
- Do not use the integer divisor value of 1 when in Fractional mode.

The following figure shows the timing for a fractional baud rate value of 2 and $3/64^{\text{th}}$. Exact averaging is accomplished in four Tbit cycles due to 64^{th} granularity.

Figure 183 • Example with Fractional Baud Rate of 2 and $3/64^{\text{th}}$

Bit Rate Modulation with Tbit Times



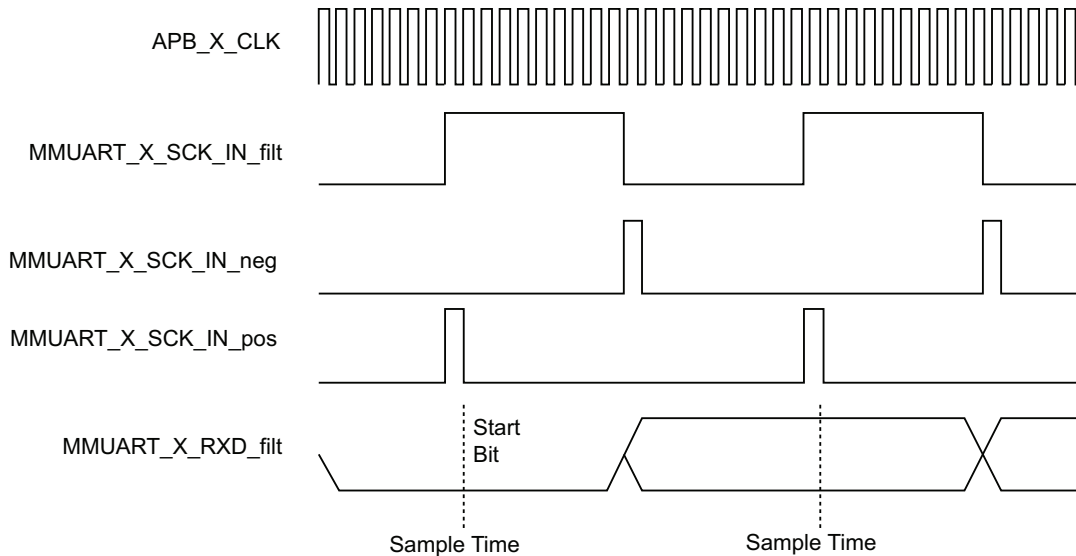
13.2.4.1.2 Input Clock-Synchronous-Slave Mode

MMUART supports a synchronous Master and Slave mode. In the synchronous Slave mode, the MMUART accepts an input clock that is synchronized to the data. The asynchronous state machine is harnessed to self-synchronize the start bit to a clock edge. With this circuit, the input clock can be

positive-edge or negative-edge; the first edge that detects a start bit aligns the state machine to sample data on the next same edge, as shown in the following figure.

In the synchronous Slave mode, the Tx block does not output a clock or baud rate signal as it is assumed that the master provides clocking for any other slave peripherals. Additionally, the APB clock (APB_X_CLK) frequency must be at least 2x the input MMUART_X_SCK_IN (serial input synchronous clock) since MMUART_X_SCK_IN gets resynchronized.

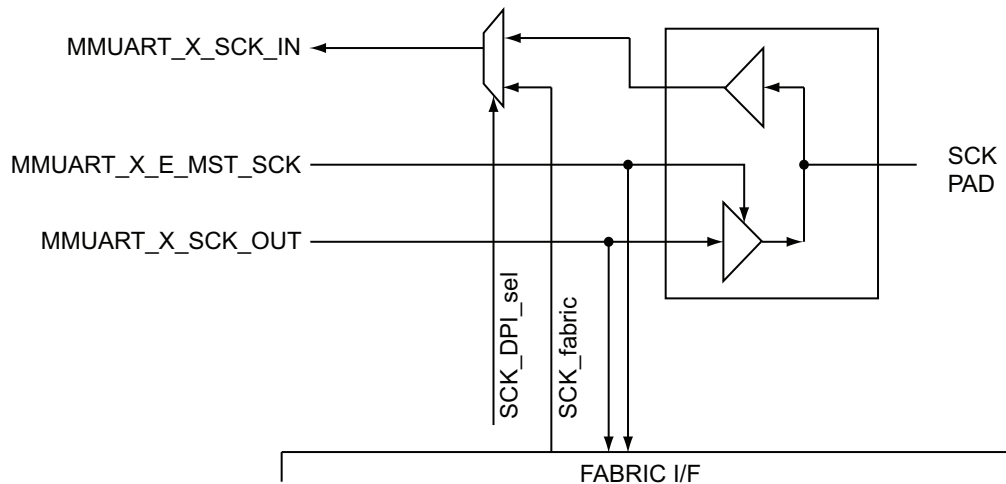
Figure 184 • Synchronous Input and Adaptation to Internal Baud Clocking



13.2.4.1.3 Clock Out-Synchronous-Master Mode

The synchronous Master mode is similar to the synchronous Slave mode except that an external clock source (MMUART_X_SCK_IN), the receiver and transmitter blocks use an internally generated synchronous clock which must be no greater than $\frac{1}{2}$ APB clock frequency. The output clock, MMUART_X_SCK_OUT, is provided to the slave devices. The internally generated synchronous clock is set with the same baud rate divisor registers that are used in the Asynchronous mode. The Fractional baud rate generation mode should not be used in the Synchronous mode as the output clock can become unpredictable.

The MMUART_X_E_MST_SCK, master clock output enable (Table 464, page 471) is used for controlling a bi-directional pad that is used for MMUART_X_SCK_IN and MMUART_X_SCK_OUT. MMUART_X_E_MST_SCK=1 indicates Master mode, forcing a bi-directional pad to be an output, otherwise the pad acts as input for MMUART_X_SCK_IN. MMUART_X_E_MST_SCK=1 indicates Master mode, forcing a bi-directional pad to be an output, otherwise the pad acts as input for MMUART_X_SCK_IN.

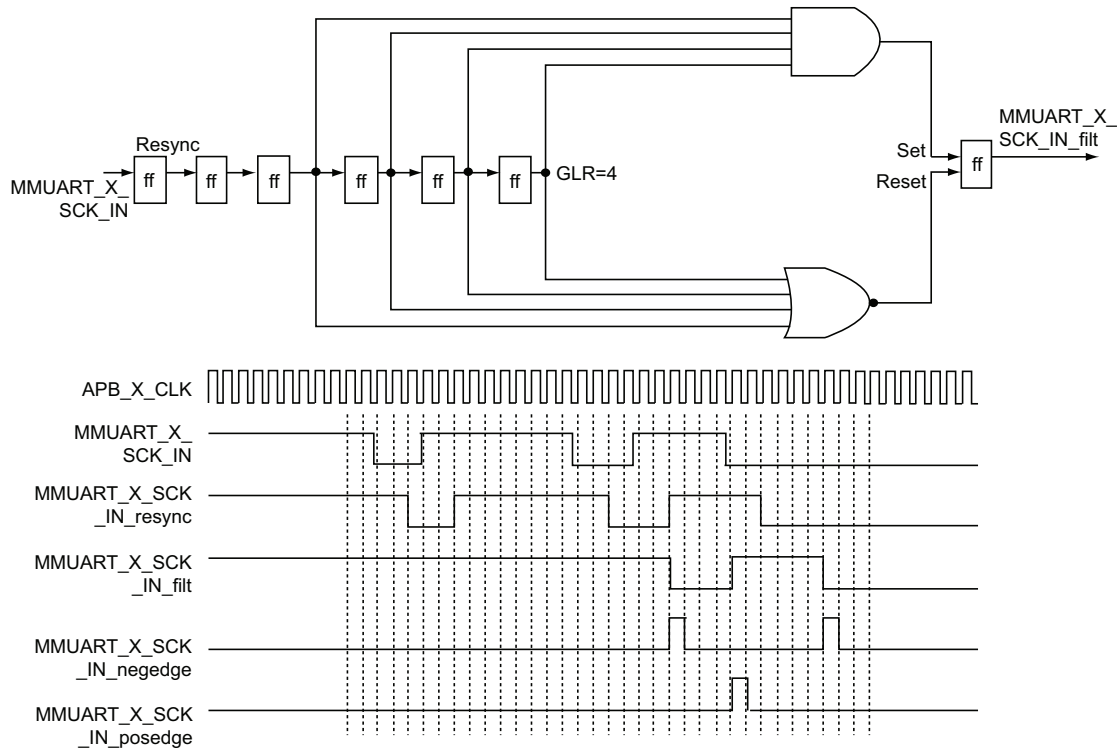
Figure 185 • Bi-Directional Synchronous Clock Configuration Options

13.2.4.2 Input Filters

MMUART provides input filters for general purposes and suppression of noise and spikes. After resynchronization, the input filters utilize all-zero/all-one unanimous sampling technique based on the system clock with a configurable filter length of N FFs, set by the GLR bits of the [GFR](#) Register. If all N FFs are 0, then 0 is set on the output. Similarly, if all N FFs are 1, then 1 is set on the output. The input filters may be bypassed by setting GLR to 0 in which case only two resynchronize flip-flops are used. Setting the GLR value to 1 adds a metastability flip-flop and provides no spike filtering. GLR values 2 to 7 further filter the spikes in the output received from flip-flops. Thus, this method helps to suppress spikes for GLR width greater than 1 APB clock cycle.

An example is shown in the following figure with GLR = 4. In all cases, positive edge and negative edge signals are generated on the same cycle as the positive or negative edge of the resynchronized and/or filtered output signal. Note that in order to obtain the maximum rate for synchronous UART operation (2x APB_X_CLK), set N to zero.

Filtering in Synchronous mode is still possible, but the sampling rate would have to be decreased to less than the filter length.

Figure 186 • Input Filtering Circuit and Timing for GLR=4 (Pulses Less than 4 APB Clock Cycles Filtered Out)


13.2.4.3 LIN Header Detection and Auto Baud Rate Calculation

The LIN is a serial communications protocol, which efficiently supports the control of mechatronics nodes in distributed automotive applications. LIN is a broadcast serial network, comprising of one master and one or more slaves.

In a LIN system, the Master and Slave nodes are initially set to a given baud rate between 1 and 20 Kbps. The master controls the per-frame-change of the asynchronous baud rate with the LIN header's break/sync fields. The communication in an active LIN network is always initiated by the master task, as shown in the following figure. The master sends out a message header, which comprises the break field, synchronization byte field, followed by a protected identifier field (PID).

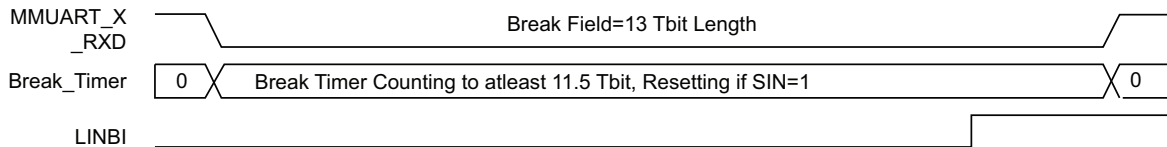
Figure 187 • LIN Header


13.2.4.3.1 Break/Sync Detection

When the LIN header detection block is enabled with the ELIN bit (MM0), the LIN circuit automatically detects break fields greater than 11 Tbits (bit time) of the currently running baud rate. The LIN circuit calculates the number of APB clock cycles from the 1st sync byte falling edge to the fifth (and last) sync byte falling edge with a 23-bit counter. The 23-bit counter is necessary to count eight Tbits at the slowest baud rate. Once the 5th edge is detected, the LIN circuit automatically updates the integer and fractional divisor registers and the PID and rest of the frame can be handled with the normal UART Rx/Tx paths. Every LIN frame begins with the break, which comprises of break lengths greater than 11 Tbits.

This serves as a start-of-frame notice to all nodes on the bus. The break field signals the start of a new frame, as shown in the following figure.

Figure 188 • LIN Break Field Width => 11 Tbit Count Interrupt

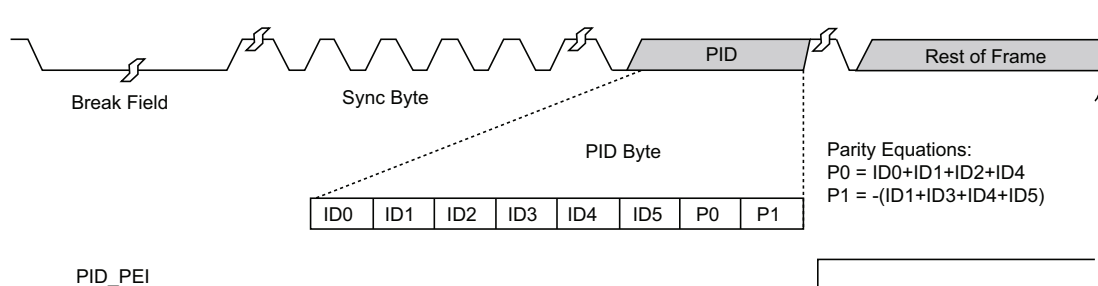


The sync field is the second field transmitted by the master task in the header. Sync is defined as the character x55. The sync field allows slave devices that perform automatic baud rate detection to measure the period of the baud rate and adjust their internal baud rates to synchronize with the bus. The sync field allows for fine calibration of the nominal baud rate, as shown in the following figure. After the LINBI bit is set in multi-mode interrupt register (IIM), the first falling edge resets the sync timer and the fifth falling edge stops counting. The 23-bit count value is divided by 128 (128 = 16 baud clocks per bit X 8 bits) to generate the integer baud rate by shifting the count value by seven towards LSB. The fractional value is contained in the remaining bits, using the first insignificant bit to round the result.

The PID field is the final field transmitted by the master task in the header. This field provides identification for each message on the network and ultimately determines which nodes in the network receive or respond to each transmission. All slave tasks continually listen for PID fields, verify their parities, and determine if they have to receive data or send data for this particular identifier. The LINSI interrupt resets the FIFO address pointers so that the PID stays in the first location. The firmware reading the PID byte determines, if the application needs to send or receive data, and the trigger level that needs to be set. A PID parity error check is performed on the fly and the interrupt, PID_PEI, is asserted when there is a mismatch between the incoming P0 and P1 parity bits and the calculated P0 and P1 parity.

The PID field shown in the following figure is processed as the first byte and placed in the receive FIFO. The parity is calculated as per the equations shown, and an interrupt (PID_PEI) is generated, if there is a mismatch between the calculated PID parity and the received PID parity.

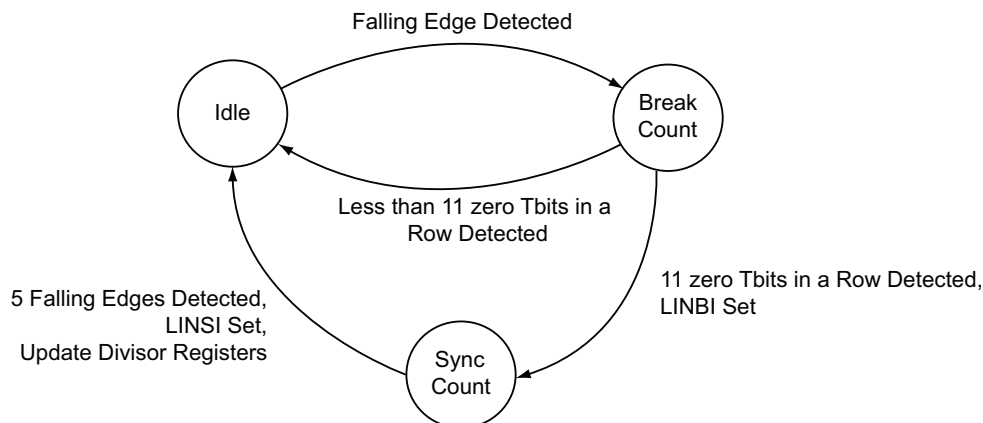
Figure 189 • LIN PID Parity Error Interrupt



13.2.4.3.2 Auto Baud Rate Update FSM

The following figure describes the receiving LIN FSM. The FSM simply follows through by parsing the break and sync fields, and then auto-updating the integer and fractional baud rate divisor register before returning to idle.

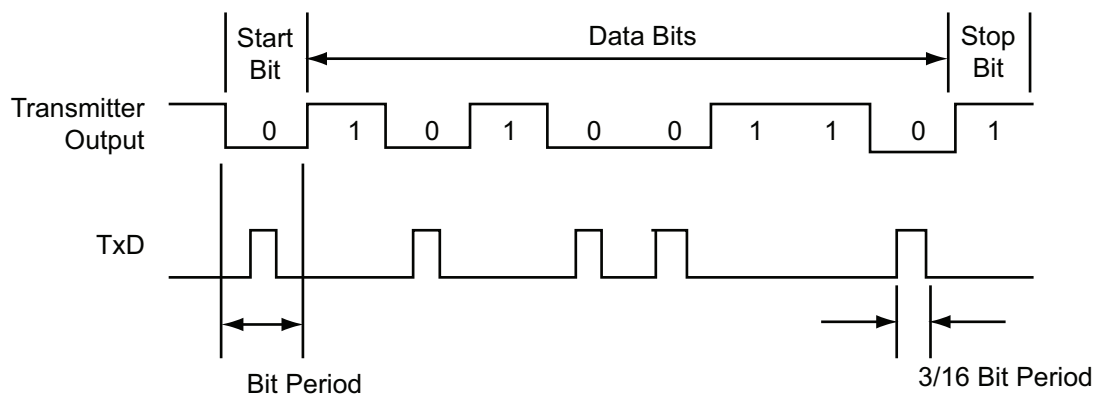
Figure 190 • LIN Receive FSM



13.2.4.4 RZI Modulation and Demodulation

The RZI modulation/demodulation blocks are intended to allow for IrDA serial infrared (SIR) communications. The SIR specification uses RZI signaling for improving the peak power to average power ratio in low rate wireless applications such as remote controls. RZI modulation scheme, a logic 0 is represented as an infrared light pulse, as shown in the following figure. A logic zero is represented as a $3/16^{\text{th}}$ high pulse, while no pulse in a given baud rate bit time equates to logic 1. UART 16x over-sampling, start and stops bits are used. Modifications to a standard UART consist of adding an IrDA modulator and demodulator in the TxD and RxD signal paths.

Figure 191 • RZI Modulation



RZI demodulation takes pulses that are $3/16^{\text{th}}$ of a baud rate clock long and transforms it to the standard non-return to zero (NRZ) UART signal, which is then fed into the main UART Rx blocks. Similarly, RZI modulation takes the outgoing UART NRZ Tx signal and creates $3/16^{\text{th}}$ pulses out.

13.2.4.4.1 RZI to NRZ Demodulation

The RZI to NRZ demodulator looks for High to Low transitions on the input, marking each one as an RZI pulse for applications such as IrDA. As High to Low transitions are detected, it is important to have any noise suppressed before entering the demodulator. The input filter block (refer to the [Input Filters](#), page 479) of the MMUART is used to suppress noise.

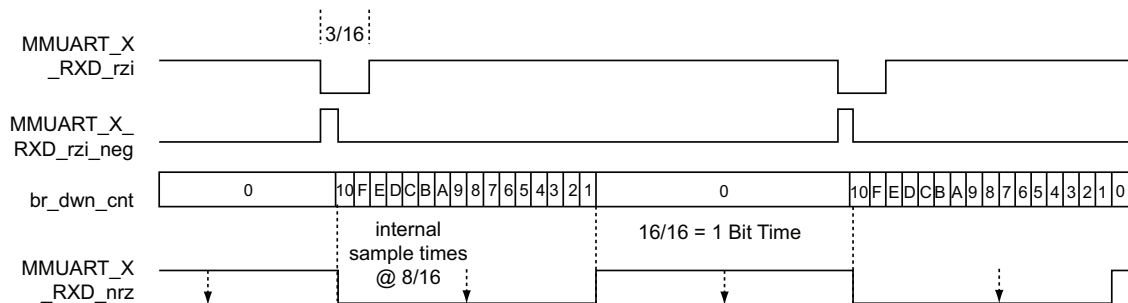
However, true IrDA compliant optical transceivers should perform noise suppression and supply the MMUART with noise free digital signals as per the IrDA specification.

After input filtering, the demodulator waits for a positive edge and sets a down counter based on the internal 16 samples per bit time baud rate clock. The output NRZ signal is set Low until the counter reaches 0, then automatically it goes High.

When back-to-back pulses occur, the incoming pulse may not be aligned with the internal baud count enable and as such may generate a 1/16th NRZ 1 level. This 1/16th pulse is a do not care situation because the sampling occurs in the middle of the Tbit time (bit time).

The input to the demodulator can be an inverted version of the following figure using the EIRX configuration bit in the multi-mode control register 1 (MM1). Also the EIRD bit in MM1 has to be set to 1.

Figure 192 • RX RZI-to-NRZ Demodulation

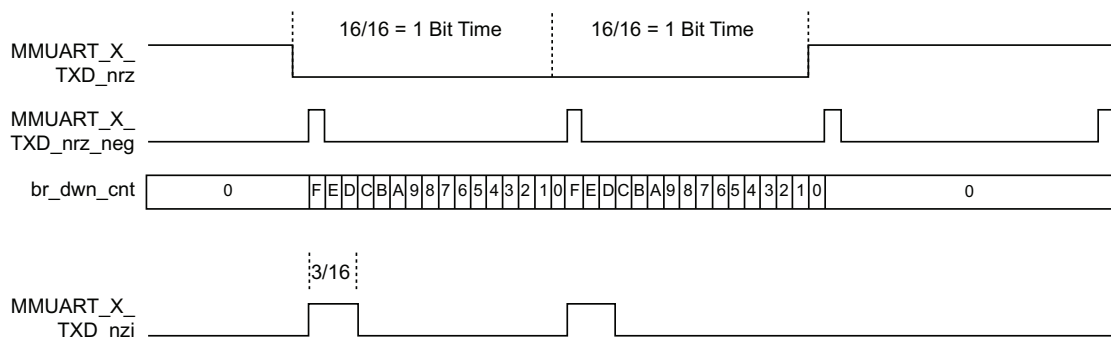


13.2.4.4.2 NRZ to RZI Modulation

The NRZ to RZI modulation receives the MMUART_X_TXD_nrz signal which is in NRZ format and needs to be converted into the RZI format. The modulator waits for a negative edge on the NRZ signal, sets a baud rate/16 decremented down counter and outputs a pulse for the first 3 or 4 counts. If the count reaches zero and the NRZ signal is still Low, another output pulse is generated, otherwise, no pulse is generated.

The output of the modulator can be an inverted version of the following figure with the EITX configuration bit in the MM1. Additionally, the output can be 1/4 widths instead of 3/16th using the EITP bit. Also the EIRD (enable bit RZI mod/demod) bit in MM1 has to be enabled. The following example shows an output for a signal that has two consecutive zeros and the value of EIRD=1, EITX=1, and EITP=0.

Figure 193 • Tx NRZ-to-RZI Modulation



Multi-Mode Config Bits in this example: EIRD = 1, EITX = 1, EITP = 0

13.2.4.5 Transmitter Time Guard

When the transmitter time guard (TTG) feature is enabled and configured for a time value greater than 0, it allows programmable wait periods between transmissions. It can be configured in the TTG register. On each byte transfer, the transmitter enters a TTG wait state prior to sending out the stop bits. Tx Time Guard Value = TTG x Bit Time (Tbit)

13.2.4.6 Receiver Timeout

In the Receiver timeout block (Rx timeout), a counter looks for the post filtered serial input to remain High until the counter expires. Once the timeout occurs, an RTOII interrupt is generated that is cleared when the **RTO** register is rewritten.

Rx Timeout Value = 4 x RTO x Bit Time (Tbit)

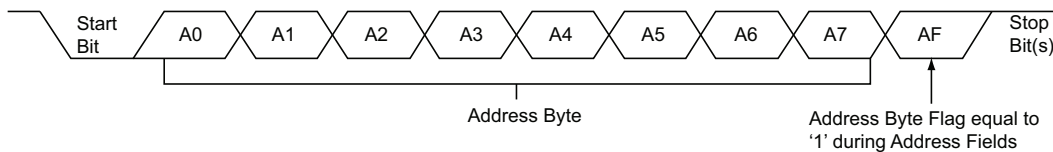
13.2.4.7 LSB/MSB Orientation

MMUART_x allows either LSB or MSB to be received or transmitted first. LSB/MSB orientation is mapped based on the Tx or Rx MSB bits of the **MM1**. Remapping of the serial I/O occurs at the APB interface.

13.2.4.8 9-bit Transmit/Receive Mode

MMUART_x supports 9-bit address flag capability. The 9th bit mode is required for multi-drop addressing topologies. Addressing mode is essentially 8-bit mode with 1 parity error bit or the 9th bit with parity set at stick 0, which means that the parity is permanently set to 0 while receiving or transmitting data, an error condition of 1 is used as an address flag, as shown in the following figure.

Figure 194 • 9-Bit Format



There are two ways to handle 9-bit multi-drop addressing:

- Software Driven Parity Error Checking:** This is accomplished by configuring the MMUART in 8-bit mode, with stick 0 parity (SP bit in **LCR**) enabled. A parity error (PE) in the line status register (**LSR**) is set whenever an address flag (AF) arrives by marking the address. Software can then check this address byte to see if it matches its own address, and then proceed accordingly.
- Automatic Hardware Address Flag Comparison:** When the automatic address flag comparison option is enabled with the EAFM bit in the Multi-mode register 2 (**MM2**), the MMUART initially disables the Rx FIFO and continuously checks for the address flag. If an address flag is received and the associated 8-bit data matches the address in the **ADR** register, the Rx FIFO is enabled. In this mode, the software does not check the address, and only receives Rx data once the address is matched. Disabling the Rx FIFO occurs either by address flag being re-sent with a non-matching address value (automatic), or the EAFC bit in **MM2** is set. An example for using EAFC takes place when the address flag is received with the correct address. If the frame length is known to be 4 bytes, then the software could set the EAFC bit to disable the Rx FIFO after the 4th received byte, and begin searching for another address flag with matching address.

13.2.4.9 Loopback Modes

There are two loopback modes: Local loopback mode and Remote loopback mode. In the Local loopback mode, MMUART_X_TXD is set to 1. The MMUART_X_RXD, MMUART_X_DSR, MMUART_X_CTS, MMUART_X_RI, and MMUART_X_DCD inputs are disconnected. The output of the transmitter shift register is looped back into the receiver shift register. The modem control outputs (MMUART_X_DTR, MMUART_X_RTS, MMUART_X_OUT1, and MMUART_X_OUT2) are connected internally to the modem control inputs, and the modem control output pins are set as 1. The transmitted data is immediately received, allowing the Cortex-M3 processor to check the operation of the MMUART_X.

In Remote loopback mode, when a bit is received, it is sent directly out of the transmit line, bypassing the transmitter block, and disabling the receiver. Local loopback has a higher priority over Remote loopback. In Automatic echo mode, a bit is received and is sent directly out the transmit line, bypassing the transmitter block, while the receiver is still enabled. Loopback modes can be enabled or disabled by making changes to the modem control register (**MCR**).

13.2.4.10 ISO7816-3 Modes

The ISO7816 is an international standard related to electronic identification cards. The ISO7816-3 utilizes a half-duplex, bi-directional bus for data transfers. The MMUART provides the clock to the smart card IC and is therefore considered the master in the system and smart card is the slave. MMUART supports T0/T1 addressing modes. Protocol T=0 is the asynchronous half-duplex character transmission protocol whereas T=1 is the asynchronous half-duplex block transmission protocol. The T0 protocol is a byte-oriented protocol where a character is transmitted across the channel between the reader (MMUART) and the card.

In addition, error handling is performed on each byte by looking at the parity bit. If the actual parity bit does not correspond to the parity of the transmitted data, then an error must have occurred. In the T0 protocol, the receiving side signals that it requires the byte to be retransmitted in case of detecting a parity error. This is done by holding the I/O line low (the I/O line is set High preceding the transfer of a byte). When the transmitting side detects this, it resends the byte that was not received correctly. The transmitter output enable is enabled during the transmission of the start bit and data byte and is disabled during the stop bit.

For T = 0 protocol, the format of the data is composed of 1 start bit, 8 data bits, 1 parity bit, and 1 guard time, which is composed of 2-bit times. The transmitter shifts all the data out except during the guard time. The guard time is used by the receiver to NACK the transmission. When the EERR bit in MM2 is set, the receiver will force an error signal to transmit out, if an incoming parity error is detected. In this case the I/O signal is held Low for one Tbit time starting from 10.5. When transmitting, the 11th Tbit in the ACK/NACK window is sampled from the Rx input, generating an interrupt if NACK (NACKI interrupt can be enabled by modifying IEM register) is detected. The following figures show the timing diagrams for I/O signals, the Transmit mode output enable and the Receive mode output enable when EERR is set to 1.

Note: The Tx Transmit mode output enable (TE) is disabled during a parity error to prevent any wrong data from being transmitted.

Similarly, the Rx mode output enable (TE) is enabled during parity error to allow the retransmission of data.

Figure 195 • Single Wire Error Signal Timing when EERR=1

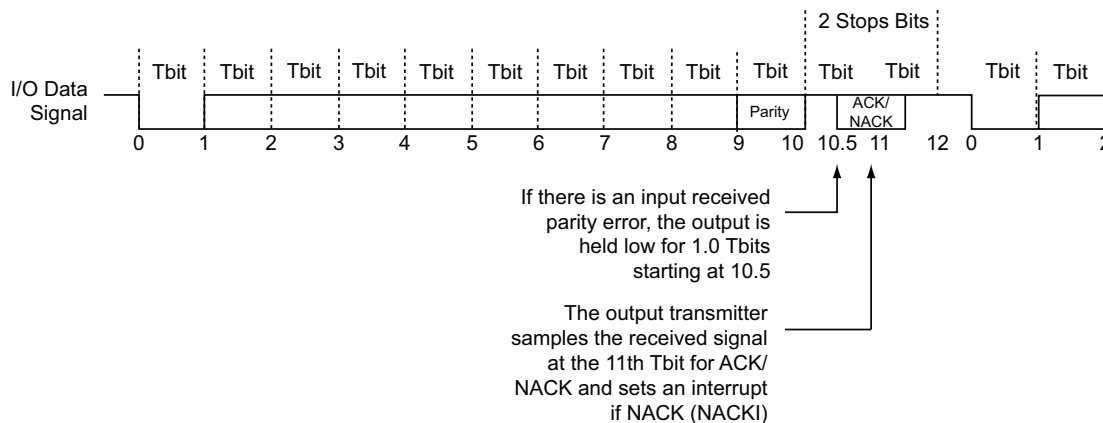


Figure 196 • Transmit Mode TE Output Enable Timing when EERR=1

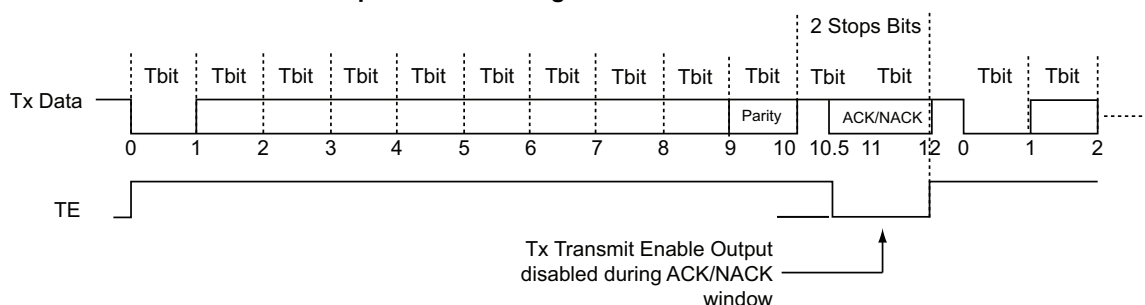
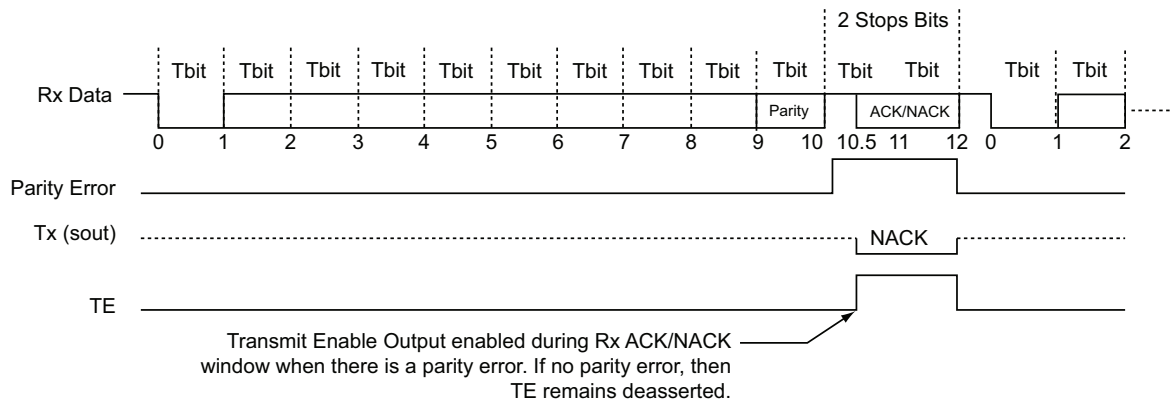


Figure 197 • Receive Mode TE Output Enable and NACK Timing when EERR=1

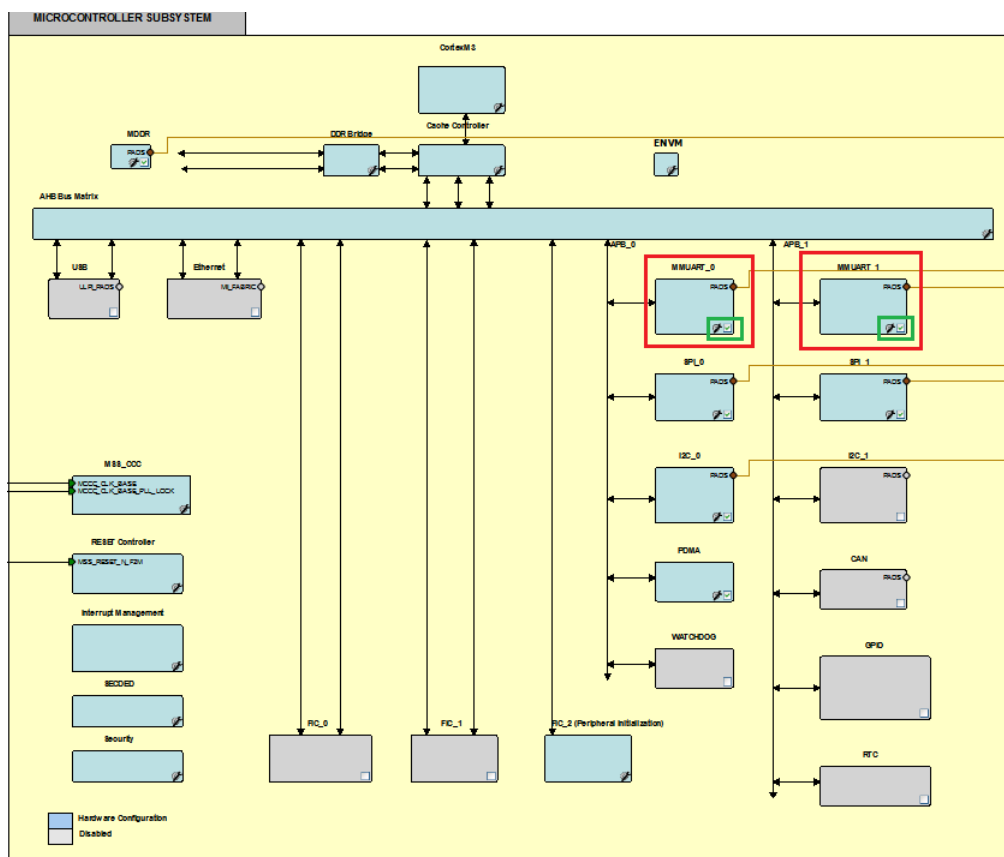
13.3 How to Use MMUART

This section describes how to use MMUART in an application.

13.3.1 Design Flow

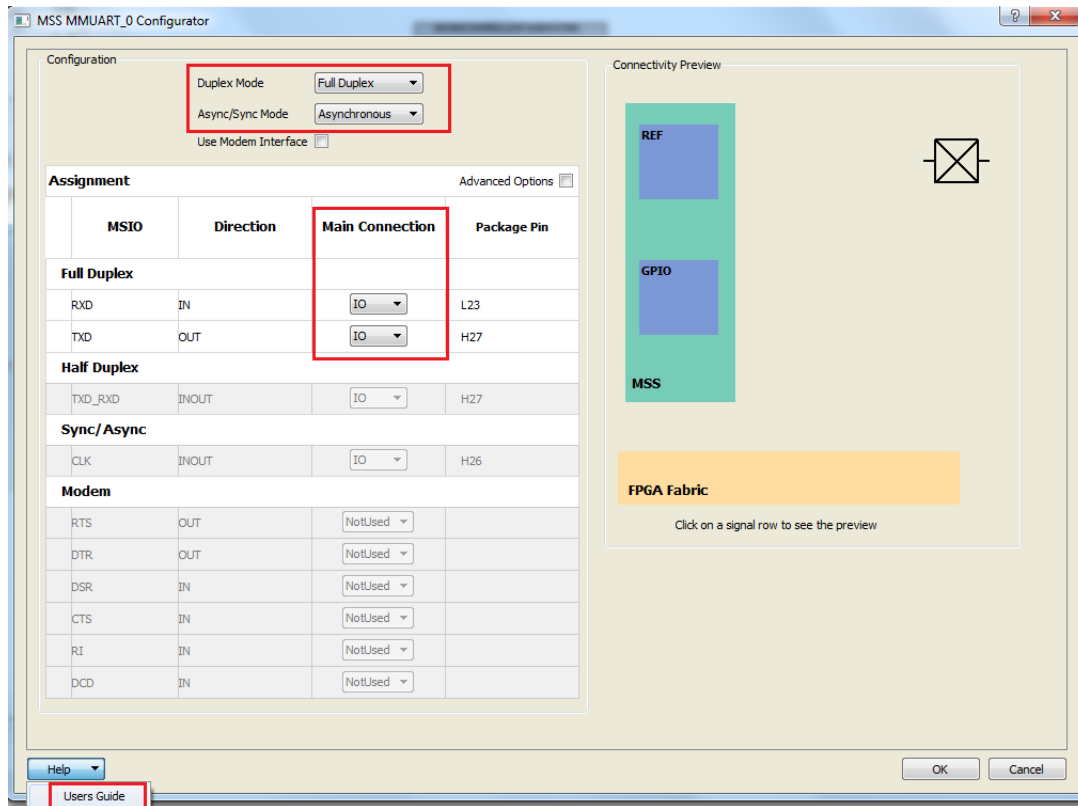
The following steps are used to enable the MMUART in the application by using Libero SoC.

1. Enable MMUART_0 and/or MMUART_1 instance by using the MSS configurator in the application, as shown in the following figure.

Figure 198 • Enable MMUART

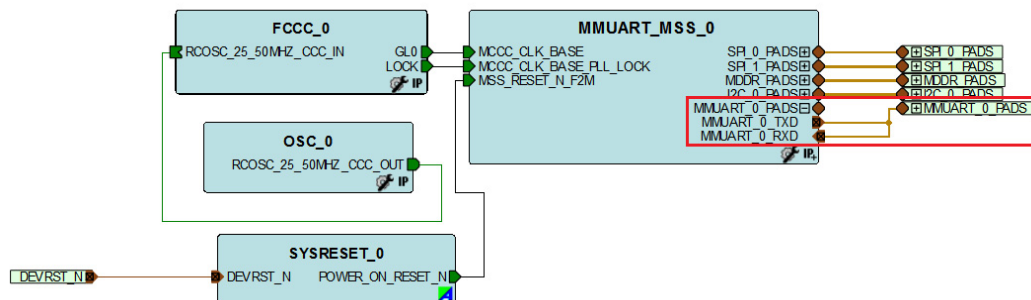
- Configure **Duplex Mode** to Full Duplex and **Async/Sync Mode** to Asynchronous by using MSS MMUART_0 Configurator as shown in the following figure. Use the **Main Connection** drop-down list to connect the ports of enabled MMUART_0 instance to an I/O. Click the highlighted **Users Guide** button to find more information on MMUART configuration details.

Figure 199 • MSS MMUART Configurator



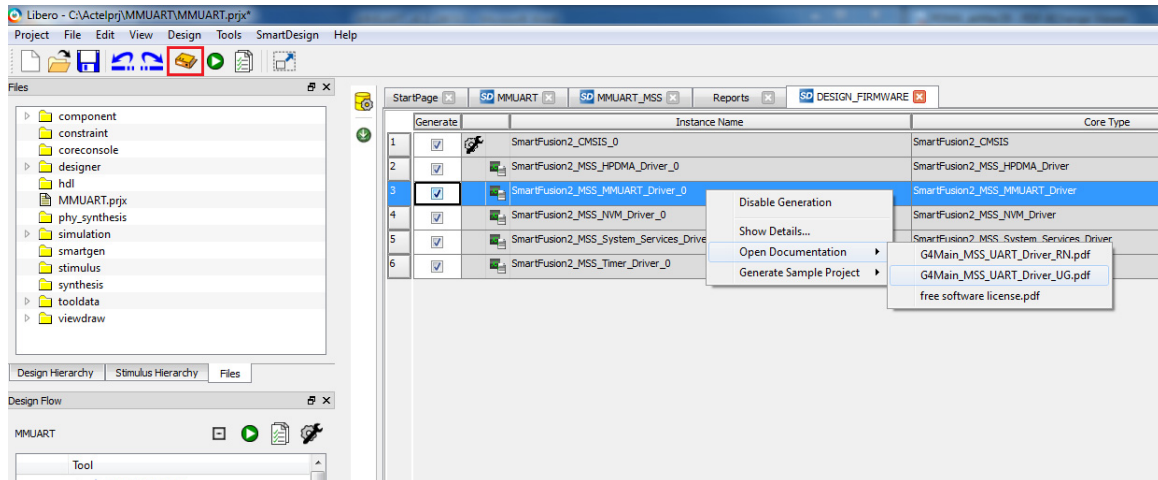
- The MMUART_0 interface signals in the MSS component are shown in the following figure.

Figure 200 • MMUART Interface Signals



4. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component**. For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace are included into the project. Click the highlighted **Configure firmware** as shown in the following figure to find the MMUART driver information.

Figure 201 • MMUART Driver User Guide



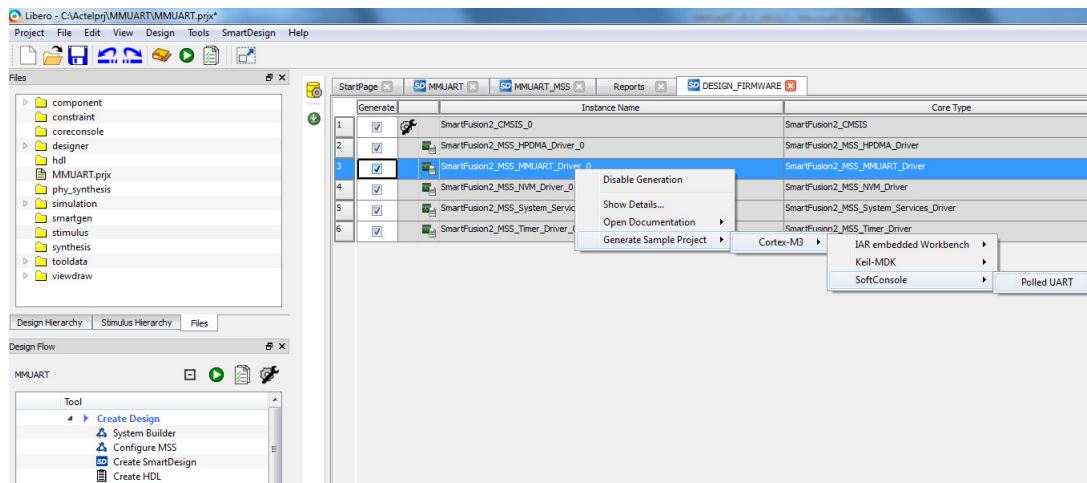
5. Click **Generate Bitstream** under **Program Design** to complete the .fdb file generation.
6. Double-click **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_uart firmware driver. The firmware driver, mss_uart (mss_uart.c and mss_uart.h), which provides a set of functions for controlling the MSS MMUARTs can also be downloaded from the Microsemi Firmware Catalog. The following table lists main APIs for MMUART. For complete information on the APIs, refer to the **SmartFusion2 MSS UART Driver User Guide** as shown in the preceding figure.

Table 466 • MSS MMUART APIs

Category	API	Description
Initialization and configuration functions	MSS_UART_init	Initializes and configures the MMUART.
	MSS_UART_lin_init	Initializes and configures the MMUART for LIN mode of operation.
	MSS_UART_irda_init	Initializes and configures the MMUART for IrDA mode of operation.
	MSS_UART_smartcard_init	Initializes and configures the MMUART for ISO 7816 (smart-card) mode of operation.
Polled transmit and receive functions	MSS_UART_polled_tx	Transmits the data.
	MSS_UART_polled_tx_string	Transmits a NULL ('\0') terminated string.
	MSS_UART_fill_tx_fifo	Fills the UART's hardware transmitter FIFO.
Interrupt driven transmit and receive functions	MSS_UART_set_rx_handler	Registers a receive handler function.
	MSS_UART_set_tx_handler	Registers a transmit handler function.
	MSS_UART_irq_tx	Initiates an interrupt driven transmission.
	MSS_UART_get_rx	Reads the content of the UART receiver's FIFO and stores it in the receive buffer

7. For more information on MMUART usage, the sample projects are available and can be generated as shown in the following figure.

Figure 202 • MMUART Sample Project



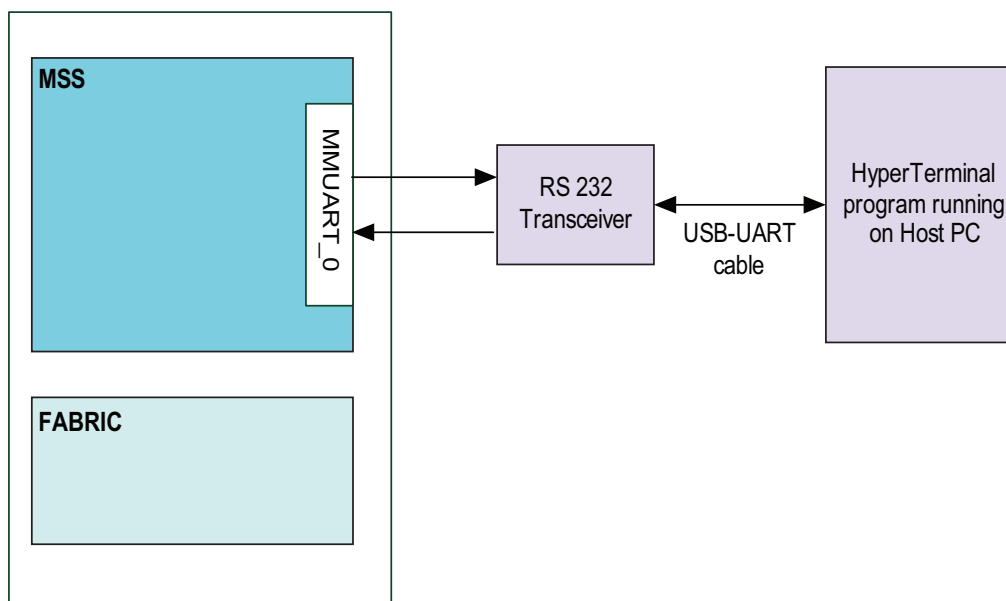
Note: The MSS MMUART does not support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for more information.

13.3.2 MMUART Use Models

13.3.2.1 Use Model: Communicating with Host PC through MMUART Peripheral Interface

This use model explains the configuration of MSS MMUART to communicate with the Host PC Hyper-Terminal program.

Figure 203 • Setup to Communicate With Host PC Through MMUART Interface - Block Diagram



Follow [Design Flow](#), page 486 to configure MMUART_0 in the application.

13.3.2.1.1 Software Design Flow

This section explains MMUART initialization and data transfers between MMUART peripheral and Host PC.

Initialization of MMUART Peripheral

Initialize the MMUART instance MMUART_0 by using MSS_UART_init API. Specify the baud rate and line configuration information like bit length, parity, and stop bits to configure the MMUART instance. The same baud rate and line information should be used to configure the Host PC HyperTerminal program.

Data Transfer

Use MSS_UART_polled_tx API with MMUART_0 instance and data buffer to send the data to the Host PC. Use MSS_UART_get_rx API with MMUART_0 instance and data buffer to receive the data from the Host PC.

13.4 MMUART Register Map

The MMUART_0 base address resides at 0x40000000 and extends to address 0x40000FFF in the Cortex-M3 processor memory map. The MMUART_1 base address resides at 0x40010000 and extends to address 0x40010FFF in the Cortex-M3 processor memory map. The following table summarizes the control and status registers for MMUART_0 and MMUART_1.

Table 467 • MMUART Register Definitions

Register Name	Divisor Latch Access Bit (DLAB) ¹	Address Offset	Read/Write	Reset Value	Description
RBR	0	0x0	R	N/A	Receiver buffer register
THR	0	0x0	W	N/A	Transmit holding register
DLR	1	0x0	R/W	0x01	Divisor latch (LSB)
DMR	1	0x04	R/W	0	Divisor latch (MSB)
DFR	N/A	0x3C	R/W	0	Fractional divisor register
IER	0	0x04	R/W	0	Interrupt enable register
IEM	N/A	0x24	R/W	0	Multi-mode interrupt enable register
IIR	N/A	0x08	R	0x01	Interrupt identification register
IIM	N/A	0x28	Clear on R	0	Multi-mode interrupt identification register
FCR	N/A	0x08	W	0	FIFO control register
LCR	N/A	0x0C	R/W	0	Line control register
MCR	N/A	0x10	R/W	0	Modem control register
LSR	N/A	0x14	R	0x60	Line status register
MSR	N/A	0x18	R	0	Modem status register
SR	N/A	0x1C	R/W	0	Scratch register
MM0	N/A	0x30	R/W	0	Multi-mode control register0
MM1	N/A	0x34	R/W	0	Multi-mode control register1
MM2	N/A	0x38	R/W	0	Multi-mode control register2
GFR	N/A	0x44	R/W	0	Glitch filter register
TTG	N/A	0x48	R/W	0	Transmitter time guard register
RTO	N/A	0x4C	R/W	0	Receiver time-out register
ADR	N/A	0x50	R/W	0	Address register

1. DLAB is the MSB of the line control register (LCR bit 7).

The following tables provide the register bit descriptions in detail.

13.4.1 Receiver Buffer Register (RBR)

Table 468 • RBR

Bit Number	Name	R/W	Reset Value	Description
[7:0]	RBR	R	N/A	This register holds the receive data bits for MMUART_x. The default value is unknown since the register is loaded with data in the receive FIFO. Bit 0 is the LSB and it is the first bit received. It may be configured as the MSB by configuring the E_MSB_RX bit in the MM1. The divisor latch access bit (DLAB), bit 7 of LCR, must be 0 to read this register. This register is read only. Writing to this register with the DLAB 0 changes the transmit holding register (THR) register value.

13.4.2 Transmit Holding Register (THR)

Table 469 • THR

Bit Number	Name	R/W	Reset Value	Description
[7:0]	THR	W	N/A	This register holds the data bits to be transmitted. Bit 0 is the LSB and is transmitted first. The MSB may be transmitted first, if it is configured with the E_MSB_TX bit in the MM1. The reset value is unknown since the register is loaded with data in the transmit FIFO. The DLAB, bit 7 of LCR, must be 0 to write to this register. This register is write only. Reading from this register with the DLAB 0 reads the RBR register value.

13.4.3 FIFO Control Register (FCR)

Rx and Tx FIFOs are 16 bytes deep.

Table 470 • FCR

Bit Number	Name	R/W	Default State	Description
[7:6]	RX_TRIG	W	0b11	These bits are used to set the trigger level for the Rx FIFO interrupt. Rx FIFO trigger level (bytes) are: 0b00: 1 byte 0b01: 4 bytes 0b10: 8 bytes 0b11: 14 bytes
[5:4]	Reserved	W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
3	ENABLE_TXRDY_RXRDY	W	0	Software must always set this bit to 1 for efficient data transfer from transmit FIFO to PDMA.
2	CLEAR_TX_FIFO	W	0	Clears all bytes in the Tx FIFO and resets its counter logic. The shift register is not cleared. 0: Disabled (default) 1: Enabled

Table 470 • FCR (continued)

Bit Number	Name	R/W	Default State	Description
1	CLEAR_RX_FIFO	W	0	Clears all bytes in Rx FIFO and resets counter logic. This shift register is not cleared. 0: Disabled (default) 1: Enabled
0	ENABLE_TX_RX_FIFO	W	1	It enables both the Tx and Rx FIFOs and is hardwired to 1, which means it is always enabled and cannot be changed.

13.4.4 Baud Rate Divisor Registers

Baud rate clock can be generated in three modes: Asynchronous mode, Synchronous Master mode, and Synchronous Slave mode. Apart from this, MMUART has an option of generating a fractional baud rate that provides more precision. For more details refer to the [Baud Rate Generation](#), page 474.

13.4.4.1 Asynchronous/Synchronous Mode Context

In Asynchronous mode, the baud rate (BR) clock is generated by dividing the input reference clock's frequency (APB_0_CLK for MMUART_0 and APB_1_CLK for MMUART_1) by 16 and the integer plus fractional divisor value, as shown below.

$$\text{BaudRate} = \frac{\text{APB_X_CLK}}{16\text{DivisorValue}}$$

Divisor Value = Integer Value (DMR + DLR registers) plus Fractional Value (DFR/64).

In the Synchronous-Master mode, the baud rate clock is generated by dividing the input reference clock's frequency (APB_0_CLK for MMUART_0 and APB_1_CLK for MMUART_1) by 2 and the integer divisor value, as shown below.

$$\text{BaudRate} = \frac{\text{APB_X_CLK}}{2\text{DivisorValue}}$$

DivisorValue = Integer Value (DMR + DLR registers).

In the Synchronous Slave mode, the baud rate clock is generated directly from the input clock, and as such the divisor registers are not used.

Note: The maximum input clock frequency is 1/2 APB_X_CLK (master clock) frequency, if input filtering is not used. If input filtering is used, then the maximum input clock frequency is determined by the following equation:

$$\text{MaxBaudRate} = \frac{\text{APB_X_CLK}}{2 + \text{GlitchFilterLength}}$$

The glitch filter length (GLR) can be configured by setting the bits in glitch filter register ([GFR](#)).

13.4.4.2 Fractional Baud Rate Register

The baud rate divisor value is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. This is used by the baud rate generator to determine the bit period. The fractional baud rate divider enables the use of any clock with a frequency >3.6864MHz to act as input clock, while it is still possible to generate all the standard baud rates. The 16-bit integer is written to the integer baud rate register ([DLR](#)). The 6-bit fractional part is written to the fractional baud rate register ([DFR](#)). [DFRs](#) are accomplished for divisor values greater than one using time averaging of the two nearest integer value baud rates based on the fractional value in the [DLR](#):

$$\text{FractionalBaudRate} = \frac{\text{APB_X_CLK}}{16(\text{DMR} + \text{DLR} + (\text{DFR}/64))}$$

EQ 1

The special case of the integer divisor value (DLR+DMR) equal to one is not allowed in fractional mode as maximum error comes when the value is one. Therefore use integer divisor values of two or more when using fractional mode.

Baud Rate Registers (DLR, DMR, and DFR)

Table 471 • DLR

Bit Number	Name	R/W	Reset Value	Description
[7:0]	DLR	R/W	0x01	This divisor latch LSB register (DLR) holds the LSB of the integer divisor value used to calculate the baud rate. The baud rate can be calculated using EQ 1, 2, 3, or 4.

Table 472 • DMR

Bit Number	Name	R/W	Reset Value	Description
[7:0]	DMR	R/W	0x00	This divisor latch MSB register (DMR) holds the MSB of the integer divisor value used to calculate the baud rate. The baud rate can be calculated using EQ 1, 2, 3, or 4.

Table 473 • DFR

Bit Number	Name	R/W	Reset Value	Description
[5:0]	DFR	R/W	0x00	The fractional divisor register (DFR) is used to store the fractional divisor used to calculate the fractional baud rate value in $1/64^{\text{th}}$. 0x0: 0/64 0x1: 1/64 0x3F: 63/64

As explained earlier, the divisor value has an integer part and a fractional part. Calculate the 6-bit number (k) which is the fractional divisor in **DFR** register by taking the fractional part of the required baud rate divisor and multiplying it by 64 (that is, 2^n , where n is the fractional part which is 6) and adding 0.5 to account for rounding errors:

$$k = \text{integer} (\text{fractional part of divisor value} \times 2^n + 0.5)$$

For example, to generate the baud rate of 134.5 the reference clock is 18.432 MHz.

$$F_{\text{APBCLK}} = 18.432 \text{ MHz, Fractional BR} = 134.5,$$

$$\text{Hence, calculate the divisor value} = (18.432 \times 10^6) / (16 \times 134.5) = 8,565.05$$

The integer part of divisor = 8565 and fractional part of divisor = 0.05

$$\text{Therefore, the fractional part, } k = \text{integer} ((0.05 \times 64) + 0.5) = 3.7 \sim 4$$

The following table contains the list of baud rates and corresponding values of DFR and DMR+DLR registers.

Table 474 • Baud Rates and Divisor Values for the 18.432 MHz Reference Clock

Baud Rate	Divisor	DLR + DMR Integer Divisor	DFR Fractional Divisor in 64 th	Percent Error
50	23,040	23,040	0	0.00000%
75	15,360	15,360	0	0.00000%
110	10,472.72	10,472	47	0.00007%
134.5	8,565.05	8,565	4	0.00008%
150	7,680	7,680	0	0.00000%
2,000	576	576	0	0.00000%
38,400	30	30	0	0.00000%
56,000	21.57	21	37	0.03255%

13.4.5 Interrupt Enable Register (IER)

Table 475 • IER

Bit Number	Name	R/W	Reset Value	Description
[7:4]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
3	EDSSI	R/W	0	Modem status interrupt enable 0: Disabled (default) 1: Enabled
2	ELSI	R/W	0	Receiver line status interrupt enable 0: Disabled (default) 1: Enabled
1	ETBEI	R/W	0	Transmitter holding register empty interrupt enable 0: Disabled (default) 1: Enabled
0	ERBFI	R/W	0	Enables received data available interrupt 0: Disabled (default) 1: Enabled

13.4.6 Multi-Mode Interrupt Enable Register (IEM)

Table 476 • IEM

Bit Number	Name	R/W	Reset Value	Description
[7:5]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
4	ELINSI	R/W	0	Enables the LIN sync detection interrupt 0: Disabled (default) 1: Enabled

Table 476 • IEM (continued)

3	ELINBI	R/W	0	Enables LIN break interrupt 0: Disabled (default) 1: Enabled
2	EPID_PEI	R/W	0	Enables PID parity error interrupt 0: Disabled (default) 1: Enabled
1	ENACKI	R/W	0	Enables NACK interrupt 0: Disabled (default) 1: Enabled
0	ERTOI	R/W	0	Enables receiver timeout interrupt 0: Disabled (default) 1: Enabled

13.4.7 Interrupt Identification Register (IIR)

Table 477 • IIR

Bit Number	Name	R/W	Reset Value	Description
[7:6]	Mode	R	0b11	Always 0b11. Enables FIFO mode.
[5:4]	Reserved	R	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read- modify-write operation.
[3:0]	IIR	R	0b0001	<p>Interrupt identification bits.</p> <p>0b0110: Highest priority. Receiver line status interrupts due to overrun error, parity error, framing error, or break interrupt. Reading the line status register resets this interrupt.</p> <p>0b0100: Second priority. Receive data available interrupt modem status interrupt. Reading the receiver buffer register (RBR) or the FIFO drops below the trigger level resets this interrupt.</p> <p>0b1100: Second priority. Character timeout indication interrupt occurs when no characters have been read from the Rx FIFO during the last four character times and there was at least one character in it during this time. Reading the RBR resets this interrupt.</p> <p>0b0010: Third priority. Transmit holding register empty interrupt. Reading the IIR or writing to the transmit holding register (THR) resets the interrupt.</p> <p>0b0000: Fourth priority. Modem status interrupt due to clear to send, data set ready, ring indicator, or data carrier detect being asserted. Reading the modem status register resets this interrupt.</p> <p>0b0011: Fifth priority. Multi-mode interrupts can occur due to any of the interrupts mentioned in IIM. For more details refer to Table 478, page 495.</p>

Table 478 • Interrupt Identification Bit Values

IIR Value[3:0]	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0110	Highest	Receiver line status	Overrun error, parity error, framing error, or break interrupt	Reading the line status register.

Table 478 • Interrupt Identification Bit Values (continued)

0100	Second	Received data available	Receiver data available	Reading the receiver buffer register or the FIFO drops below the trigger level.
1100	Second	Character timeout indication	No characters have been read from the Rx FIFO during the last four character times and there was at least one character in it during this time.	Reading the receiver buffer register.
0010	Third	Transmitter holding register empty	Transmitter holding register empty	Reading the IIR or writing into the transmitter holding register.
0000	Fourth	Modem status	Clear to send, data set ready, ring indicator, or data carrier detect	Reading the modem status register.
0011	Fifth	Multi mode interrupt	Any of the multi-mode interrupts in the IIM register.	Refer to Table 479, page 496 (IIM).

13.4.8 Multi-Mode Interrupt Identification Register (IIM)

Table 479 • IIM

Bits	Name	R/W	Reset Value	Description
[7:5]	Reserved	Clean on R	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read- modify-write operation.
4	LINSI	Clean on R	0	LIN sync detection interrupt ID. This bit set when 5 th falling edge is detected by the sync timer. It resets the FIFO address pointers so that the PID will be in the first location. Reading the IIM register clears this interrupt.
3	LINBI	Clean on R	0	LIN break interrupt, set automatically when break length of 11.5 Tbits is detected. Reading the IIM register clears this interrupt.
2	PID_PEI	Clean on R	0	Protected identifier field (PID) parity error interrupt is generated when there is a mismatch in PID in LIN header, that is, when either the P0 or P1 bits in the incoming PID byte do not match the calculated P0 and P1 error.
1	NACKI	Clean on R	0	NACK interrupt is asserted when EERR bit is set in MM2. Reading the MM2 clears the interrupt.
0	RTOII	Clean on R	0	Receiver time-out (RTO) interrupt ID. RTO interrupt is asserted when RTO value is reached by the counter. It gets cleared when writing to the RTO register.

13.4.9 Line Control Register (LCR)

Table 480 • LCR

Bit Number	Name	R/W	Reset Value	Description
7	DLAB	R/W	0	Divisor latch access bit. Enables access to the divisor latch registers during read or write operation to address 0 and 1. 0: Disabled (default) 1: Enabled

Table 480 • LCR (continued)

Bit Number	Name	R/W	Reset Value	Description
6	SB	R/W	0	Set break. Enabling this bit sets MMUART_x_TXD to 0. This does not have any effect on transmitter logic. The break is disabled by setting the bit to 0. 0: Disabled (default) 1: Set break
5	SP	R/W	0	Stick parity 0: Disabled (default) 1: Enabled When stick parity is enabled, the parity is set according to bits [4:3] as follows: 11: 0 will be sent as a parity bit and checked when receiving. 01: 1 will be sent as a parity bit and checked when receiving.
4	EPS	R/W	0	Even parity select 0: Odd parity (default) 1: Even parity
3	PEN	R/W	0	Parity enable 0: Disabled 1: Enabled. Parity is added in transmission and checked in receiving.
2	STB	R/W	0	Number of stop bits (STB) 0: 1 stop bit (default) 1: 11/2 stop bits when WLS=00 The number of stop bits is 2 for all other cases not described above (STB=1 and WLS=01, 10, or 11).
[1:0]	WLS	R/W	0	Word length select 0b00: 5 bits (default) 0b01: 6 bits 0b10: 7 bits 0b11: 8 bits

13.4.10 Modem Control Register (MCR)

Table 481 • MCR

Bit Number	Name	R/W	Reset Value	Description
7	Reserved	R/W	0	The software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read- modify-write operation.
[6:5]	RLoop	R/W	0	Remote loopback enable bits. In the Remote loopback mode, when a bit is received, it is sent directly out the transmit line, bypassing the transmitter block, and disabling the receiver. In the Automatic echo mode, when a bit is received, it is sent directly out the transmit line, bypassing the transmitter block, while the receiver is still enabled. 00: Disabled (default) 01: Remote loopback enabled 10: Automatic echo enabled 11: Reserved Local loopback mode has priority over the remote/echo loopback modes.

Table 481 • MCR (continued)

Bit Number	Name	R/W	Reset Value	Description
4	Loop	R/W	0	In the Loopback mode, MMUART_x_TXD is set to 1. The MMUART_x_RXD, MMUART_x_DSR, MMUART_x_CTS, MMUART_x_RI, and MMUART_x_DCD inputs are disconnected. The output of the transmitter shift register is looped back into the receiver shift register. The modem control outputs (MMUART_x_DTR, MMUART_x_RTS, MMUART_x_OUT1, and MMUART_x_OUT2) are connected internally to the modem control inputs, and the modem control output pins are set as 1. The transmitted data is immediately received, allowing Cortex-M3 processor to check the operation of the MMUART_x. The interrupts are operating in the Loopback mode. 0: Disabled (default) 1: Local loopback enabled The local loopback mode has priority over the remote loopback modes. LOOPBACK is only implemented in basic UART mode. It does function in LIN IRDA or Smart card modes.
3	OUT2	R/W	0	Controls the output2 (OUT2) signal. Active Low 0: OUT2n is set to 1 (default) 1: OUT2n is set to 0
2	OUT1	R/W	0	Controls the output1 (OUT1) signal. Active Low 0: OUT1n is set to 1 (default) 1: OUT1n is set to 0
1	RTS	R/W	0	Controls the request to send (MMUART_x_RTS) signal. Active Low 0: RTSn is set to 1 (default) 1: RTSn is set to 0
0	DTR	R/W	0	Data terminal ready (MMUART_x_DTR) output. Active Low 0: DTRn output is set to 1 (default) 1: DTRn output is set to 0

13.4.11 Line Status Register (LSR)

Table 482 • LSR

Bit Number	Name	R/W	Reset Value	Description
7	FIER	R	0	This bit is set when there is at least one parity error, framing error, or break indication in the FIFO. FIER is cleared when Cortex-M3 processor reads the LSR, if there are no subsequent errors in the FIFO.
6	TEMT	R	1	Transmit empty (TEMT). This bit is set to 1 when both the transmitter FIFO and shift registers are empty.
5	THRE	R	1	Transmitter holding register empty (THRE). Indicates that the MMUART_x is ready to transmit a new data byte. THRE causes an interrupt to the Cortex-M3 processor when bit 1 (ETBEI) in the interrupt enable register is 1. This bit is set when the Tx FIFO is empty. It is cleared when at least one byte is written to the Tx FIFO.
4	BI	R	0	Break interrupt (BI). Indicates that the receive data is at 0 longer than a full word transmission time (start bit + data bits + parity + stop bits). BI is cleared when Cortex-M3 processor reads the line status register (LSR). This error is revealed to the Cortex-M3 processor when it is associated character is at the top of the FIFO. When break occurs, only one zero character is loaded into the FIFO.

Table 482 • LSR (continued)

Bit Number	Name	R/W	Reset Value	Description
3	FE	R	0	Framing error (FE). Indicates that the receive byte did not have a valid stop bit. FE is cleared when Cortex-M3 processor reads the LSR . The MMUART_x tries to resynchronize after a framing error. To do this, it assumes that the framing error was due to the next start bit, so it samples this start bit twice, and then starts receiving the data. This error is revealed to Cortex-M3 processor when it is associated character is at the top of the FIFO.
2	PE	R	0	Parity error (PE). Indicates that the receive byte had a parity error. PE is cleared when the Cortex-M3 processor reads the LSR . This error is revealed to the Cortex-M3 processor when it is associated character is at the top of the FIFO.
1	OE	R	0	Overrun error (OE). Indicates that the new byte was received before the Cortex-M3 processor reads the byte from the receive buffer, and that the earlier data byte was destroyed. OE is cleared when the Cortex-M3 processor reads the LSR . If the data continues to fill the FIFO beyond the trigger level, an overrun error occurs once the FIFO is full and the next character has been completely received in the shift register. The character in the shift register is overwritten, but it is not transferred to the FIFO.
0	DR	R	0	Data ready (DR). Indicates when a data byte is received and stored in the receive buffer or the FIFO. DR is cleared to 0 when the Cortex-M3 processor reads the data from the receive buffer or the FIFO.

13.4.12 Modem Status Register (MSR)

Table 483 • MSR

Bit Number	Name	R/W	Reset Value	Description
7	DCD	R	0	Data carrier detect (DCD) (MMUART_x_DCD). The complement of DCD input. When bit 4 of the MCR is set to 1 (loop), this bit is equivalent to OUT2 in the MCR .
6	RI	R	0	Ring indicator (RI) (MMUART_x_RI). The complement of the RI input. When bit 4 of the MCR is set to 1 (loop), this bit is equivalent to OUT1 in the MCR .
5	DSR	R	0	Data set ready (DSR) (MMUART_x_DSR). The complement of the DSR input. When bit 4 of the MCR is set to 1 (loop), this bit is equivalent to RTS in the MCR .
4	CTS	R	0	Clear to send (CTS) (MMUART_x_CTS). The complement of the CTS input. When bit 4 of the MCR is set to 1 (loop), this bit is equivalent to DTR in the MCR .
3	DDCD	R	0	Delta data carrier detect (DDCD) indicator. Indicates that DCD input has changed state. Whenever bit 0, 1, 2, or 3 is set to 1, a modem status interrupt is generated.
2	TERI	R	0	Trailing edge of ring indicator (TERI) detector. Indicates that RI input has changed from 0 to 1. Whenever bit 0, 1, 2, or 3 is set to 1, a modem status interrupt is generated.

Table 483 • MSR (continued)

Bit Number	Name	R/W	Reset Value	Description
1	DDSR	R	0	Delta data set ready (DDSR) indicator. Indicates that the DSRn input has changed state since the last time it was read by the Cortex-M3 processor. Whenever bit 0, 1, 2, or 3 is set to 1, a modem status interrupt is generated.
0	DCTS	R	0	Delta clear to send (DCTS) indicator. Indicates that the CTSn input has changed state since the last time it was read by the Cortex-M3 processor. Whenever bit 0, 1, 2, or 3 is set to 1, a modem status interrupt is generated.

13.4.13 Scratch Register (SR)

Table 484 • SR

Bit Number	Name	R/W	Reset Value	Description
[7:0]	SCR	R/W	0	Scratch register. This register has no effect on MMUART_x operation.

13.4.14 Multi-Mode Control Register 0 (MM0)

Table 485 • MM0

Bit Number	Name	R/W	Default State	Description
7	EFBR	R/W	0	Enable fractional baud rate (FBR) mode. 0: Disabled (default) 1: Enabled
6	ERTO	R/W	0	Enable receiver timeout (RTO). Writing this bit enables the timeout and restarts the counter value. The timeout value is determined by the RTO register. 0: Disabled (default) 1: Enabled
5	ETTG	R/W	0	Enable transmitter time guard (TTG). The time guard value is determined by the TTG Register. 0: Disabled (default) 1: Enabled
4	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read- modify-write operation
3	ELIN	R/W	0	Enable LIN header detection and automatic baud rate calculation. 0: Disabled (default) 1: Enabled
[2:0]	ESYN	R/W	0	Enable synchronous operation. There are four types of Synchronous Operation modes that can be enabled. 0b000: Disabled, that is, Asynchronous mode (default) 0b001: Synchronous slave enabled, positive-edge clock 0b010: Synchronous slave enabled, negative-edge clock 0b011: Synchronous master enabled, positive-edge clock 0b100: Synchronous master enabled, negative-edge clock 0b101, 0b110, and 0b111: Reserved

13.4.15 Multi-Mode Control Register 1 (MM1)

Table 486 • MM1

Bit Number	Name	R/W	Default State	Description
[7:6]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read- modify-write operation.
5	EITP	R/W	0	Output pulse width for RZI mod can be modified using this bit. 0: 3/16 th Tbit pulse width (default). 1: 1/4 th Tbit pulse width.
4	EITX	R/W	0	You can configure output polarity for RZI modulation. 0: RZI output pulses are active Low and signify a low NRZ value (default). 1: RZI output pulses are active High and signify a high NRZ value.
3	EIRX	R/W	0	You can configure input polarity for RZI demodulation. 0: RZI input pulses are active Low, signifying a low NRZ value (default). 1: RZI input pulses are active High, signifying a high NRZ value.
2	EIRD	R/W	0	Enables RZI modulation/demodulation. 0: Disabled (default) 1: Enabled
1	E_MSB_TX	R/W	0	LSB or MSB can be sent first by configuring this bit. By default, the "THR" bit 0 is the LSB, and is the first transmitted bit. Bit 0 of the THR may be configured as the last transmitted bit, MSB. 0: THR's bit 0 is the first transmitted bit, LSB (default). 1: THR's bit 0 is the last transmitted bit, MSB.
0	E_MSB_RX	R/W	0	LSB or MSB can be received first by configuring this bit. By default, the receiver buffer register's (RBR) bit 0 is the LSB, and is the first received bit. Bit 0 of the RBR may be configured as the last received bit, MSB. 0: RBR's bit 0 is the first received bit, LSB (default). 1: RBR's bit 0 is the last received bit, MSB.

13.4.16 Multi-Mode Control Register 2 (MM2)

Table 487 • MM2

Bit Number	Name	R/W	Reset Value	Description
[7:4]	Reserved	R/W	0	The software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read- modify-write operation.
3	ESWM	R/W	0	Enable single-wire, half-duplex mode. 0: Disabled (default) 1: Enabled
2	EAFM	R/W	0	Enable a flag clear (EAFM). When EAFM is enabled the Rx FIFO is disabled until another address flag with matching address is received. The bit gets cleared on write in multi-mode control registers 2. 0: Disabled (default) 1: Enabled

Table 487 • MM2 (continued)

Bit Number	Name	R/W	Reset Value	Description
1	EAFM	R/W	0	<p>Enable automatic 9-bit address flag mode (EAFM). It should be noted that for enabling this bit it requires, the LCR should be in an 8-bit and stick parity (SP) bit configured to 0. If EAFM bit is disabled, the Rx FIFO is enabled by receiving all the bytes. When EAFM bit is enabled, the Rx FIFO is disabled until an address flag with matching address is received. If an address match occurs and the Rx FIFO is enabled then it can be disabled, if either another address flag occurs or there is a mismatch or the EAFM bit is set. In either case, the Address flag compare will continue as long as the EAFM bit is set.</p> <p>0: Disabled (default) 1: Enabled</p>
0	EERR	R/W	0	<p>When the EERR bit is set, the receiver forces an error signal transmit out, if an incoming parity error is detected. Error signal (ACK/NACK) is sent during stop time enable.</p> <p>The EERR only applies in an 8-bit data length, 2 stop bit configuration. Error signal occurs during the last 1.5 stop bits as per Figure 195, page 485.</p> <p>0: Disabled (default) 1: Enabled</p>

13.4.17 Glitch Filter Register (GFR)

Table 488 • GFR

Bit Number	Name	R/W	Reset Value	Description
[7:3]	Reserved	R/W	Reserved	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read- modify-write operation.
[2:0]	GLR	R/W	0	<p>The glitch filter resynchronizes (GLR) and suppresses random input noise from MMUART_x_RXD (serial input data) and MMUART_x_SCK_IN (serial input clock in synchronous mode) based on the filter length given in number system clock cycles. The following are the different filter lengths in the APB clock cycles that can be written into the GLR register and their description.</p> <p>0b000: Two resynchronize flip-flops are used but there is no spike suppression.</p> <p>0b001: Three resynchronize flip-flops are used but there is no spike suppression.</p> <p>0b010: Three resynchronize flip-flops are used and it also causes 1 APB clock cycle suppression.</p> <p>0b011: Three resynchronize flip-flops are used and it also causes 2 APB clock cycle suppression.</p> <p>0b100: Three resynchronize flip-flops are used and it also causes 3 APB clock cycle suppression.</p> <p>0b101: Three resynchronize flip-flops are used and it also causes 4 APB clock cycle suppression.</p> <p>0b110: Three resynchronize flip-flops are used and it also causes 5 APB clock cycle suppression.</p> <p>0b111: Three resynchronize flip-flops are used and it also causes 6 APB clock cycle suppression.</p>

13.4.18 Transmitter Time Guard Register (TTG)

Table 489 • TTG

Bit Number	Name	R/W	Reset Value	Description
[7:0]	TTG	R/W	0	If the transmitter time guard is enabled from the multi-mode control register 0 (MM0), the transmitter time guard value determines the amount of system clock cycles to wait between transmissions. The time guard equation is based on the baud rate bit time (Tbit) value as follows: Tx Time Guard Value = TTG x Bit Time (Tbit)

13.4.19 Receiver Timeout Register (RTO)

Table 490 • RTO

Bit Number	Name	R/W	Reset Value	Description
[7:0]	RTO	R/W	0	Writing to the RTO register sets the counter value and enables, if the ERT0 bit in the MM0 is enabled. You can configure the timeout value by writing into this register. The RTO counts when the Rx block input state is idle; is reset when a start condition occurs, and restarts counting upon returning to the idle state. When the RTO value is reached, the RTOII interrupt is set. Re-writing the RTO register clears the interrupt and sets the counter. The receiver timeout value equation is based on the baud rate bit time (Tbit) as follows: Rx Timeout Value = 4 x RTO x Bit Time (Tbit)

13.4.20 Address Register (ADR)

Table 491 • ADR

Bit Number	Name	R/W	Reset Value	Description
[7:0]	ADR	R/W	0	The address register is used in 9-bit Address Flag mode. When an address flag is received on the 9 th bit, and EAFM is set in MM2, the incoming data is checked against the address register. If a match occurs, the Rx FIFO is enabled.

14 Serial Peripheral Interface Controller

Serial peripheral interface (SPI) is a synchronous serial data protocol that enables the microprocessor or microcontroller and peripheral devices to communicate with each other. The SPI controller is an APB slave in the SmartFusion2 device that provides a serial interface compliant with the Motorola SPI, Texas Instruments synchronous serial, and National Semiconductor MICROWIRE™ formats. In addition, SPI supports interfacing with large SPI flash and EEPROM devices and a hardware-based slave protocol engine.

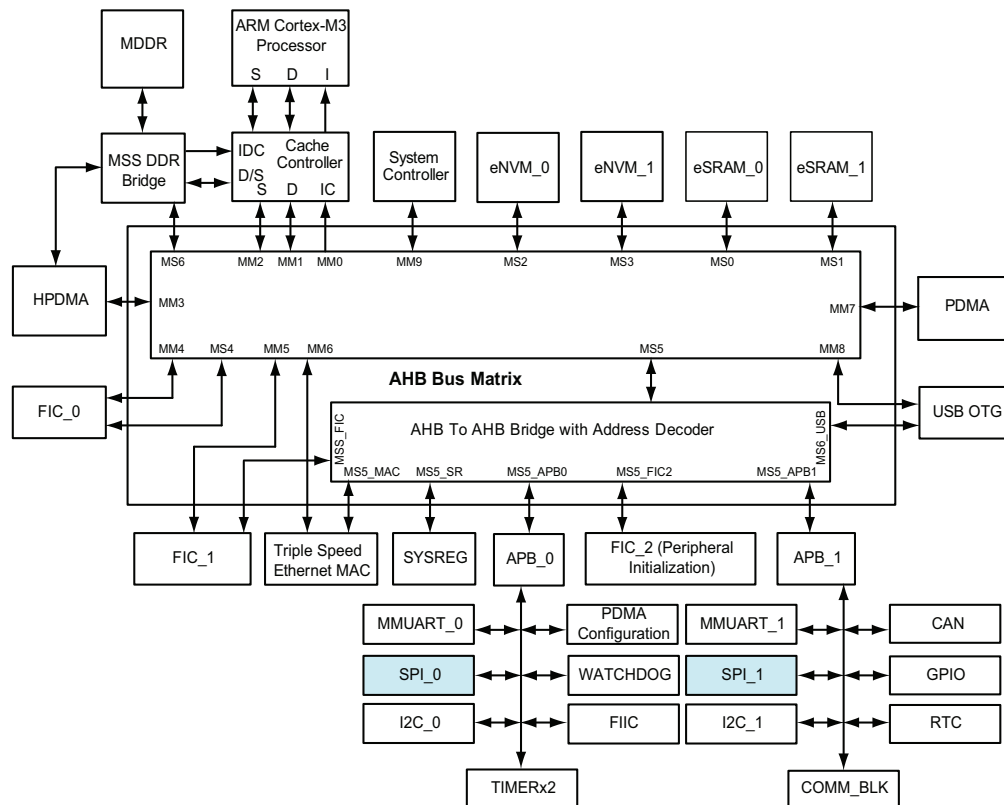
14.1 Features

SmartFusion2 SPI peripherals support the following features:

- Master and Slave modes
- Selectable slaves up to 8
- Configurable slave select operation
- Configurable clock polarity
- Separate transmit (Tx) and receive (Rx) FIFOs to reduce interrupt service loading
- Processor controlled and PDMA controlled mode of data transfer

The following figure shows details of the microcontroller subsystem (MSS). The SPI peripherals are interfaced to the AHB bus matrix through the APB interfaces (APB_0 and APB_1).

Figure 204 • Microcontroller Subsystem Showing SPI Peripherals



14.2 Functional Description

This section provides the detailed description of SPI peripherals.

14.2.1 Architecture Overview

The SPI controller supports master and slave modes of an operation.

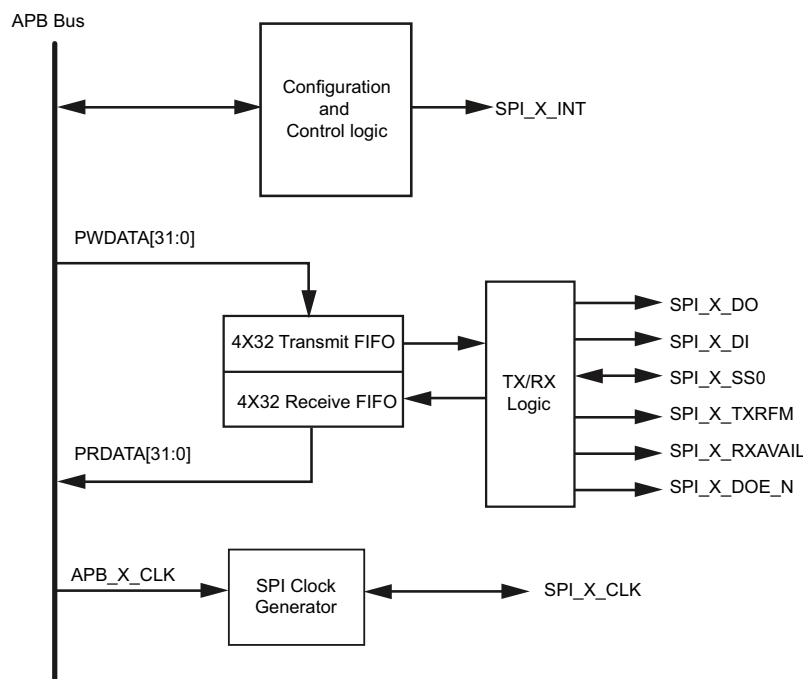
- In master mode, the SPI generates SPI_X_CLK, selects a slave using SPI_X_SS[x], transmits the data on SPI_X_DO, and receives the data on SPI_X_DI.
- In slave mode, the SPI is selected by SPI_X_SS[0]. The SPI receives a clock on SPI_X_CLK and incoming data on SPI_X_DI.

The SPI peripherals consist mainly of the following components (refer [Figure 205](#), page 505).

- Transmit and receive FIFOs
- Configuration and control logic
- SPI clock generator

The following figure shows the SPI controller block diagram.

Figure 205 • SPI Controller Block Diagram



Note:

- The SPI_X_DO, SPI_X_DI, SPI_X_SS0, and SPI_X_CLK signals are available to the FPGA fabric.
- SPI_X_DOE_N is accessible through the SPI control register.
- SPI_X_TXRFM and SPI_X_RXAVAIL signals are used only by PDMA.
- SPI_X_INT is sent to the Cortex-M3 processor.

Note: X is used as a place holder for 0 or 1 in the register and signal descriptions. It indicates SPI_0 (on the APB_0 bus) or SPI_1 (on the APB_1 bus).

14.2.1.1 Transmit and Receive FIFOs

The SPI controller embeds two 4 × 32 (depth × width) FIFOs for receive and transmit, as shown in [Figure 205](#), page 505. These FIFOs are accessible through RX data and TX data registers (refer to the [SPI Register Details](#), page 528). Writing to the TX data register causes the data to be written to the transmit FIFO. This is emptied by the transmit logic. Similarly, reading from the RX data register causes the data to be read from the receive FIFO. The not-empty port of the receive FIFO and the not-full port of the transmit FIFO flags (of the FIFOs) are exposed as SPI_X_RXAVAIL (SPI has data to be read) and

SPI_X_TXRFM (SPI has room for more data to send) ports. These are connected to the peripheral DMA (PDMA) engine to allow continuous DMA streaming for large SPI transfers and to help free up the Cortex-M3 processor.

14.2.1.2 Configuration and Control Logic

The SPI peripheral can be configured for master or slave mode by using the mode bit of the SPI **CONTROL** register. The type of data transfer protocol can be configured by using the TRANSFPRTL bit of the SPI **CONTROL** register. The control logic monitors the number of data frames to be sent/received and enables the interrupts when the data frame transmission/reception is completed. During data frames transmission/reception, if a transmit under-run error/receive overflow error is detected, the **Status** register is updated (refer to the **STAT8** register for bit definitions).

14.2.1.3 SPI Clock Generator

In master mode, the SPI clock generator generates the serial programmable clock from the APB clock. Refer to the **SPI Clock Requirements**, page 517 for more details.

14.2.2 Interface

This section provides the details of the SPI interfacing ports and various data transfer protocols.

14.2.2.1 Port List

The following table lists the SPI signals.

Table 492 • SPI Interface Signals

Name	Type	Polarity	Description
SPI_X_DI	Input	High	Serial data input
SPI_X_DO	Output	High	Serial data input
SPI_X_CLK	Input/Output	High	Serial clock. It is a serial programmable bit rate clock out signal. Input when SPI is in the slave mode. Output when SPI is in the master mode.
SPI_X_SS[0]	Input/Output	Low, except for TI mode	Slave select. Input when SPI is in the slave mode. Output when SPI is in the master mode. The slave select output polarity is active Low. In TI mode the slave select output is inverted to become active High.
SPI_X_SS[7:1]	Output	Low, except for TI mode	Extra slave select signal. Valid only in the Master mode. The slave select output polarity is active Low. In TI mode the slave select output is inverted to become active High.
SPI_X_INT	Output	High	SPI interrupt
SPI_X_DOE_N	Output	High	Output enable
SPI_X_TXRFM	Output	High	SPI ready to transmit. Used only by MSS PDMA engine
SPI_X_RXAVAIL	Output	High	SPI received data. Used only by MSS PDMA engine.

14.2.2.2 Data Transfer Protocol Details

The SmartFusion2 SPI controller supports the following data transfer protocols:

- Motorola SPI Protocol
- National Semiconductor MICROWIRE Protocol
- Texas Instruments Synchronous Serial Protocol
- Slave Protocol Engine

This section describes the data transfer protocols, timing diagrams, signal requirements, and error case scenarios for the above protocols.

14.2.2.3 Motorola SPI Protocol

The Motorola SPI is a full duplex, four-wire synchronous transfer protocol which supports programmable clock polarity (SPO) and clock phase (SPH). The state of SPO and SPH control bits decides the data transfer modes as detailed in the following table.

Table 493 • Data Transfer Modes

Data Transfer Mode	SPO	SPH
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0
Mode 3	1	1

The SPH control bit determines the clock edge that captures the data.

- When SPH is Low, data is captured on the first clock transition.
 - Data is captured on the rising edge of SPI_CLK when SPO = 0 (Figure 206, page 508).
 - Data is captured on the falling edge of SPI_CLK when SPO = 1 (Figure 209, page 509).
- When SPH is High, data is captured on the second clock transition (rising edge if SPO = 1).
 - Data is captured on the falling edge of SPI_CLK when SPO = 0 (Figure 208, page 509).
 - Data is captured on the rising edge of SPI_CLK when SPO = 1 (Figure 210, page 509).

The SPO control bit determines the polarity of the clock and SPS defines the slave select behavior.

- When SPO is Low and no data is transferred, SPI_CLK is driven to Low (Figure 207, page 508).
- When SPO is High and no data is transferred, SPI_CLK is driven to High (Figure 209, page 509).

The following table summarizes the clock active edges in various SPI master modes.

Table 494 • Summary of Master SPI Modes

Mode	SPS	SPO	SPH	Clock in Idle	Sample Edge	Shift Edge	Select in Idle	Select Between Frames
Motorola	0	0	0	Low	Rising	Falling	High	Pulses between all frames
	0	1	0	High	Falling	Rising	High	
	0	0	1	Low	Falling	Rising	High	Does not pulse between back-to-back frames. Pulses if transmit FIFO empties.
	0	1	1	High	Rising	Falling	High	Does not pulse between back-to-back frames. Pulses if transmit FIFO empties.
	1	0	0	Low	Rising	Falling	High	Stays active until all the frames set by frame counter are transmitted.
	1	0	1	Low	Falling	Rising	High	
	1	1	0	High	Falling	Rising	High	
	1	1	1	High	Rising	Falling	High	
Texas Instruments	0	0	0	Low	Falling	Rising	Low	Normal operation SPI_X_CLK only generated with select and data bits.
			1	Low	Falling	Rising	Low	Removes SPI_X_SS[0] on consecutive frames (back-to-back), making them appear to be big frames.
		1		Running	Falling	Rising	Low	SPI_X_CLK is free running.

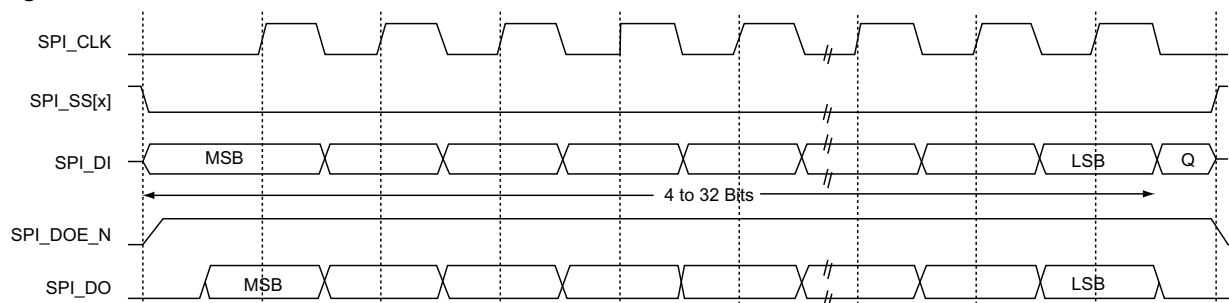
Table 494 • Summary of Master SPI Modes (continued)

Mode	SPS	SPO	SPH	Clock in Idle	Sample Edge	Shift Edge	Select in Idle	Select Between Frames
National Semiconductor Microwire	0	0	0	Low	Rising	Falling	High	Normal operation SPI_X_CLK only generated with select and data bits.
			1	Low	Rising	Falling	High	Forces IDLE cycles (SPI_X_SS[0] deactivated) between back-to-back frames.
		1		Running	Rising	Falling	High	SPI_X_CLK is free running.
	1			Low	Rising	Falling	High	After sending the command part of the frame, the subsequent frames are concatenated to create a single large data frame (master operation only).

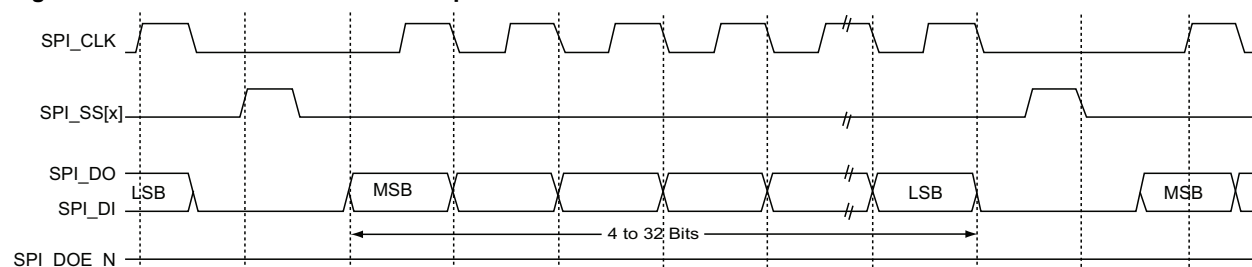
14.2.2.3.1 Motorola SPI Modes

Motorola SPI modes are shown in the following figures.

Single Frame Transfer – Mode 0: SPO = 0, SPH = 0

Figure 206 • Motorola SPI Mode 0

Multiple Frame Transfer – Mode 0: SPO = 0, SPH=0

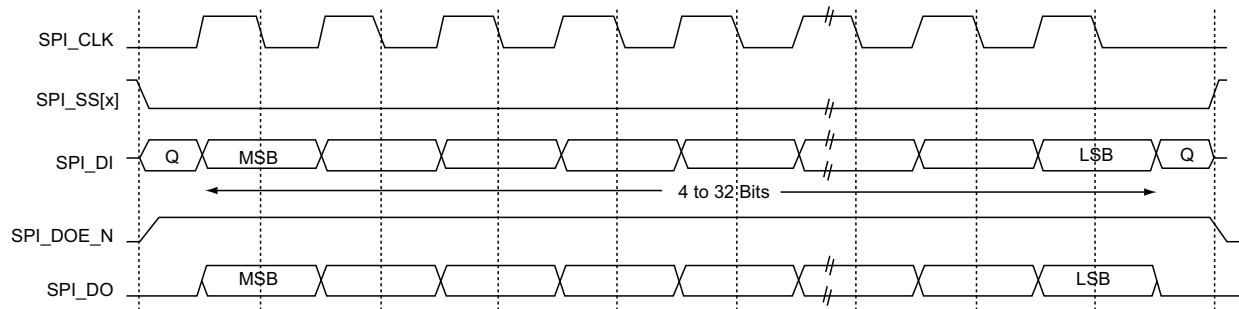
Figure 207 • Motorola SPI Mode 0 Multiple Frame Transfer

Notes:

- Between frames, the slave select (SPI_SS[x]) signal is asserted for the duration of the clock pulse.
- Between frames, the clock (SPI_CLK) is Low.
- Data is transferred to most significant bit (MSB) first.
- The output enable (SPI_DOE_N) signal is asserted during the transmission and deasserted at the end of the transfer (after the last frame is sent).

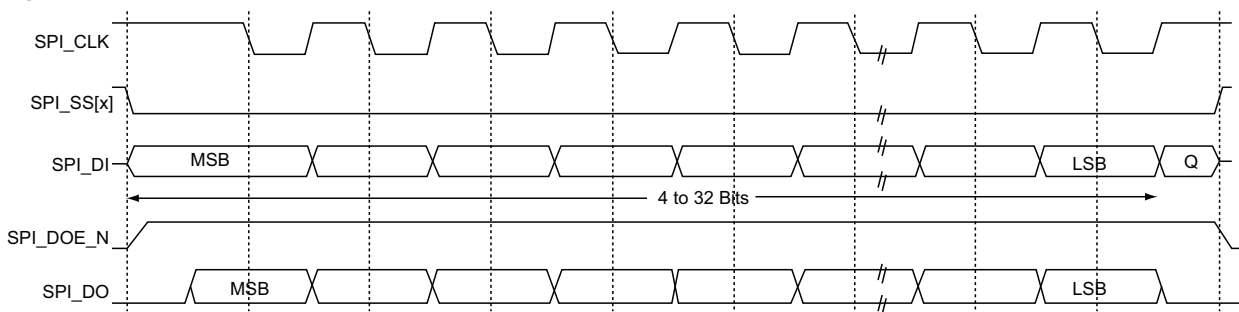
Single Frame Transfer – Mode 1: SPO = 0, SPH = 1

Figure 208 • Motorola SPI Mode 1



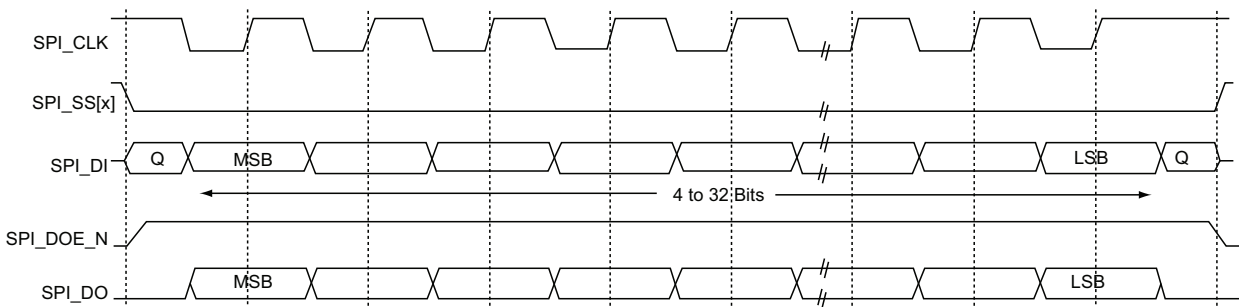
Single Frame Transfer – Mode 2: SPO = 1, SPH = 0

Figure 209 • Motorola SPI Mode 2



Single Frame Transfer – Mode 3: SPO = 1, SPH = 1

Figure 210 • Motorola SPI Mode 3



14.2.2.3.2 Output Enable (SPI_X_DOE_N) Timing

Each SPI mode comprises two phases: transmit and receive. It is a requirement that the output enable (SPI_X_DOE_N) line, which enables the output signal, should be driven so that the following occurs:

- The output signal is ready to transmit when the data is available (setup time).
- The output signal is held on long enough for the recipient to sample the data (hold time).

The minimum setup and hold time is one half SPI_X_CLK. In slave mode, the input clock is withdrawn at the end of the transfer. For example, consider the waveform for Single Frame Transfer – Mode 2: SPO = 1, SPH = 0. In this case, data is sampled on the falling edge of the clock and shifted on the rising edge of the clock. The data is sampled on the falling edge and must be held for one half SPI_X_CLK after the last

falling edge at the end of the transmission. This means that SPI_X_DOE_N must be held High for at least one half SPI_X_CLK after the last falling edge to satisfy the hold time requirement.

Table 495 • Behavior of the Output Enable Signal

Mode	Master	Slave
MOTOROLA	SPI_X_DOE_N is asserted with identical timing to that of SPI_X_SS[0]. This provides an additional half SPI_X_CLK cycle of data turn on and off relative to the data bit valid requirements.	The incoming SPI_X_SS[0] signal is used to directly generate the SPI_X_DOE_N. Similar to the master case, it provides an additional half clock cycle of data turn on and off.
Texas Instruments	SPI_X_DOE_N is asserted on the negative clock edge prior to the MSB (while SPI_X_SS[0] is asserted) and if the uninterrupted data is deasserted on the falling SPI_X_CLK edge following the LSB. This provides half a clock cycle of data turn on off time.	SPI_X_DOE_N is asserted on the positive SPI clock edge as the MSB is the output. SPI_X_DOE_N is deasserted on the positive SPI clock edge at the end of the LSB data bit, assuming no consecutive data.
National Semiconductor MICROWIRE	SPI_X_DOE_N is asserted with SPI_X_SS[0], and then removed at the start of the ninth data bit (turn around cycle).	SPI_X_DOE_N is asserted at the start of the tenth bit as data becomes valid. SPI_X_DOE_N is deasserted at the end of the LSB, if a falling clock edge occurs or when SPI_X_SS[0] is deasserted.

14.2.2.3.3 Motorola SPI Error Case Scenarios

The SPI protocol does not specify any error recovery strategy. The master and slave require prior knowledge of clock rates and data-frame layouts. However, there are built-in mechanisms in the SPI controller to recover from error. If the slave encounters an error, the master can toggle the slave clock until it comes to a known state. Here are three specific scenarios and error the behavior of the SPI controller in Motorola protocol mode.

- If the slave select signal is withdrawn in the middle of a transfer, the transfer continues until the end of the data frame.
- If the input clock is withdrawn, the SPI controller remains paused until the clock is restarted. It picks-up where it left off.
- If the slave select signal is withdrawn before a transfer occurs, the slave remains in the idle state (no data transfer having been initiated).

The SPI controller has no built-in timer. For applications where there is a possibility of a slave going to sleep for a long time, or in the case of very long transfers, the application should use a timer created from user logic.

14.2.2.3.4 SPI Data Transfer for Large Flash/EEPROM Devices in Motorola SPI Modes

Serial flash and EEPROM devices can be driven using Motorola SPI modes. Following is an outline of the interfaces to the required flash/EEPROM devices that shows how they can be driven using Motorola SPI modes. In each of these modes, the SPI controller is configured as a master with the slave select line connected to the chip select of the memory device.

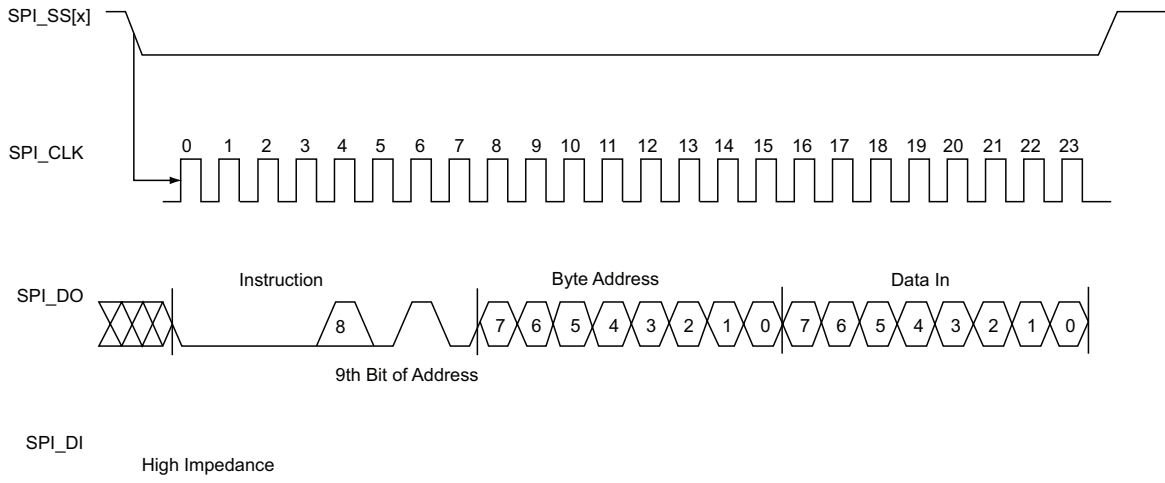
Devices Requiring Data Frame Sizes of Up to 32 Bits

Serial flash/EEPROM devices, such as the Atmel 25010/020/040, have a data frame size smaller than 32 bits and can be directly driven from SPI mode.

Write Operation for Atmel 25010/020/040 Devices

The following figure shows the write operation timing for Atmel 25010/020/040 devices. The SPI controller selects the devices using the slave select signal. The data frame size is set to 24 bits. The SPI is configured with SPO = 0, SPH = 0. The first byte is the instruction. Bit 5 of the instruction is part of the address (the 9th bit as required by the Atmel part). Bits 8-15 form a byte address. The residual 8 bits correspond to the data to be written.

Figure 211 • Write Operation Timing

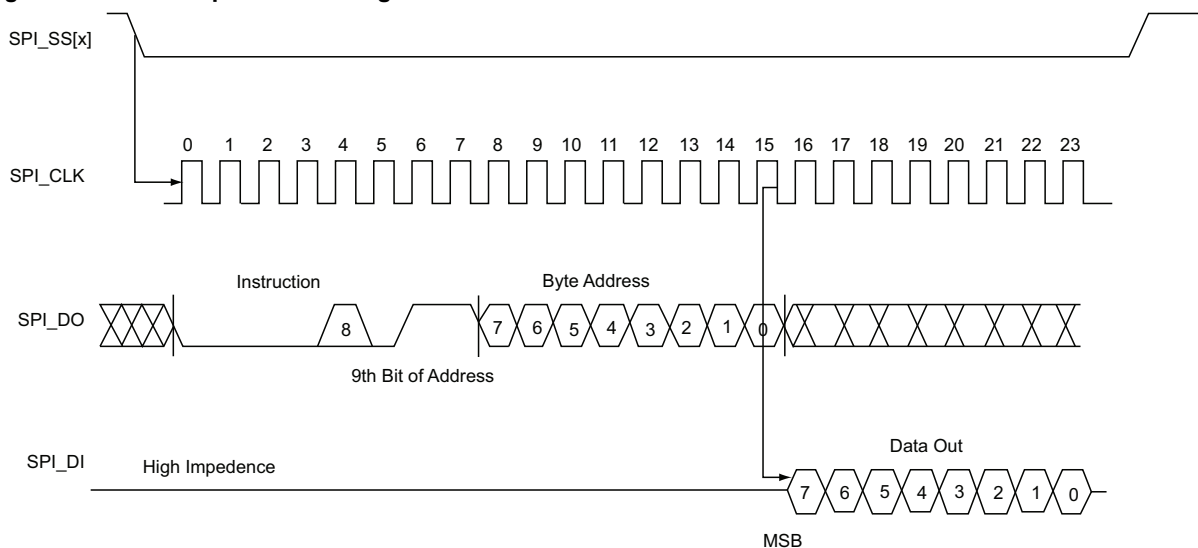


Note: The first byte contains the opcode that defines the operations to be performed. The opcode also contains address bit A8 in both the READ and WRITE instructions. This is mandated by the Atmel device.

Read Operation for Atmel 25010/020/040 Devices

The following figure shows the read operation timing for Atmel 25010/020/040 devices. For the read operation, the data frame size is set to 24 bits and the SPI controller is configured with SPO = 0, SPH = 0. On completing, the least significant byte of the received data frame corresponds to the data read.

Figure 212 • Read Operation Timing



Note: The first byte contains the opcode that defines the operations to be performed. The opcode also contains address bit A8 in both the read and write instructions. This is mandated by the Atmel device.

14.2.2.3.5 Devices Requiring Data Frame Sizes of More than 32 Bits

Serial flash devices such as the Atmel AT25DF321 which support mode 3 (SPO = 1 and SPH = 1) require more than 32 bits of frame data in some modes. To drive these devices, continuous transfers are required from the SPI interface while holding the slave select low continuously (which is connected to the chip select of the target device). This is accomplished by using the transmit FIFO from the SPI, which enforces continuous back-to-back transfers, if it is not empty. The slave select continues to be held low (active) in SPI mode 3 (SPO = 1 and SPH = 1) and not pulsed between data frames.

For example, to send 64 bits to the AT25DF321 (8-bit opcode, 24-bit address, 4 data bytes), the data frame size (TXRXDF_SIZE) can be set to 32 and the data frame count set to two (CONTROL[TXRXDFCOUNT] field).

TXRXDFCOUNT Register

The SPI peripheral contains a TXRXDFCOUNT counter (found in the CONTROL register) that counts the number of transmitted and received frames. Its function varies in master and slave modes.

TXRXDFCOUNT in master mode controls the following:

- The Tx and Rx done interrupts
- Terminates the auto fill and empty operations
- Holds slave select active

TXRXDFCOUNT in slave mode controls the following:

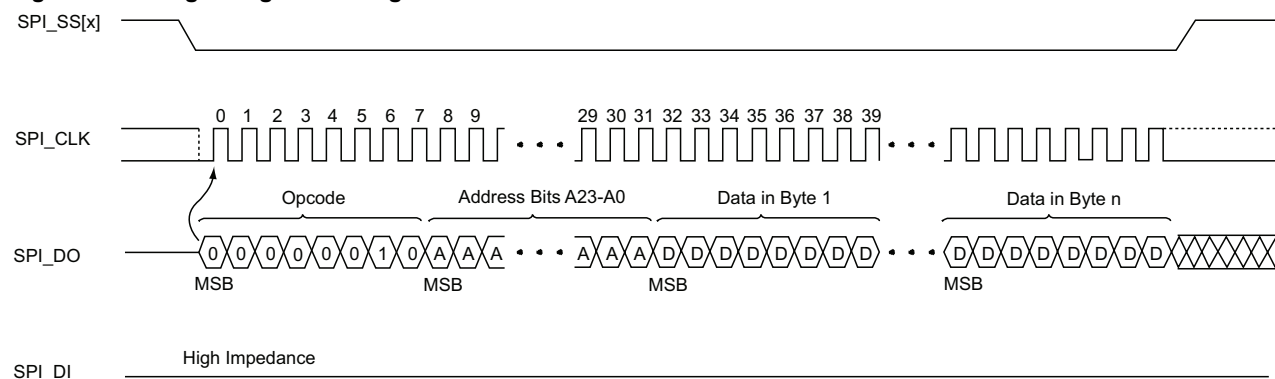
- The Tx and Rx done interrupts
- Terminates the auto fill operation

In slave operation, it is possible for TXRXDFCOUNT to miscount actual transmitted and received frames if the transmit FIFO under-run condition occurs. If this is likely in an application, Microsemi recommends that TXRXDFCOUNT not be used and that it be disabled. Instead use the CMDINT and SSEND bits in the raw interrupt status (RIS) register to monitor operation, or simply count how many frames it is received.

Page Program for Atmel AT25DF321

The following figure shows the Page Programming Timing for Atmel AT25DF321. In this mode, the opcode, address, and data require more than 32 clock periods. To drive this device, the chip select (CS) can be connected to the slave select signal, the data frame size set to 16, and the FIFO repeatedly filled until the target flash device is programmed. As long as the data is available to transmit in the FIFO, the chip select signal (connected to slave select on the SPI controller) will be asserted Low.

Figure 213 • Page Program Timing



Devices That Do Not Support Mode 1 (SPO = 0 and SPH = 1) or Mode 3 (SPO = 1 and SPH = 1)

For flash devices that do not support mode 1 (SPO = 0 and SPH = 1) or mode 3 (SPO = 1 and SPH = 1), it is necessary to use a dedicated GPIO pin to drive the chip select signal.

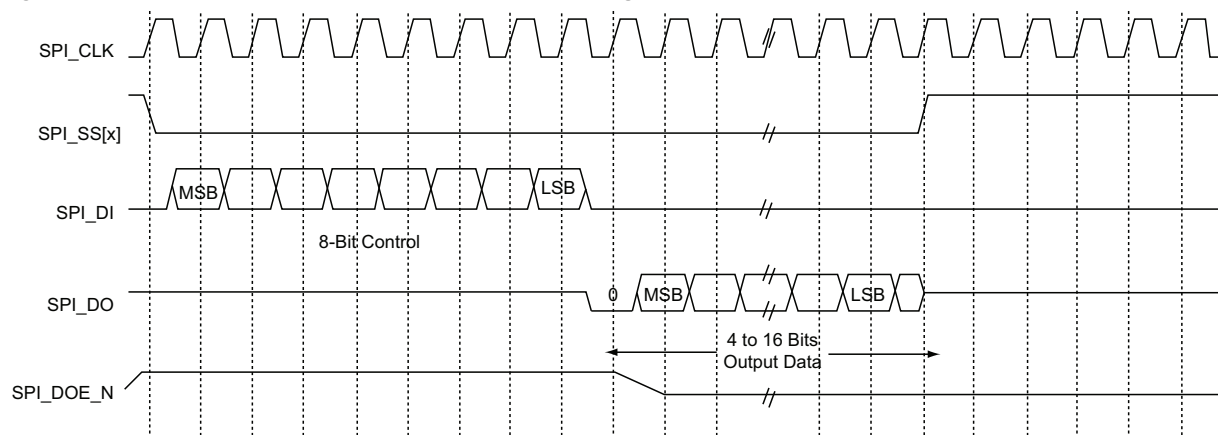
14.2.2.4 National Semiconductor MICROWIRE Protocol

The National Semiconductor MICROWIRE serial interface is a half-duplex protocol using a master/slave message passing technique. Each serial transmission begins with an 8-bit control word, during which time no incoming data is received. After the control word is sent, the external slave decodes it, and after waiting one serial clock cycle from the end of the control word, responds with the required data, which may be 4 to 16 bits in length.

14.2.2.4.1 Single Frame Transfer

In single frame transfer mode (shown in the following figure), the most significant byte of the FIFO transmit word is the control byte. The total data frame size supplied must be at least 12 bits long (8 bits for the control word and a minimum of 4 bits for data payload). Only the output data is sampled and inserted in the receive FIFO.

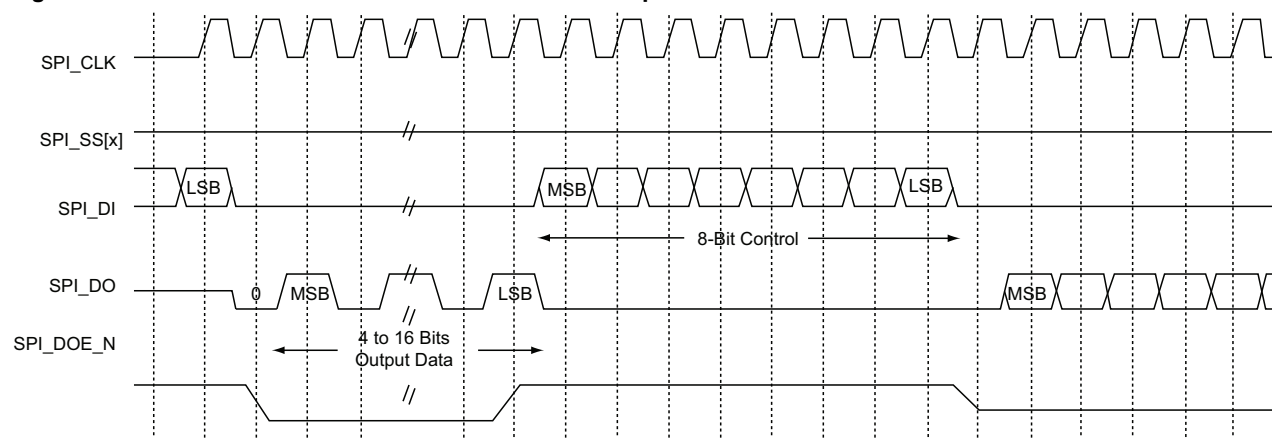
Figure 214 • National Semiconductor MICROWIRE Single Frame Transfer



14.2.2.4.2 Multiple Frame Transfer

In the multiple frame transfer (shown in the following figure), the slave select signal (SPI_X_SS[x]) is continuously asserted (held Low) while SPI_X_DOE_N (output Enable) is also asserted (or held Low) for the duration of each control byte. The other data transfers proceed in back-to-back manner.

Figure 215 • National Semiconductor MICROWIRE Multiple Frame Transfer



14.2.2.5 Texas Instruments Synchronous Serial Protocol

The Texas Instruments (TI) synchronous serial interface is based on a full duplex, four-wire synchronous transfer protocol. The transmit data pin is put in a high-impedance mode (tristated) when no data is transmitted.

- The slave select (SPI_X_SS[x]) signal is pulsed between transfers to guarantee a high-to-low transition between each frame.
- The slave select output polarity is inverted to become active High. In an idle state, the slave select (SPI_X_SS[x]) signal is kept low.
- Data is available on the clock cycle immediately following the slave select (SPI_X_SS[x]) assertion.
- Both the SPI master and the SPI slave capture each data bit into their serial shift registers on the falling edge of the clock (SPI_X_CLK). The received data is latched on the rising edge of the clock (SPI_X_CLK).
- The output enable signal (SPI_X_DOE_N) is asserted (active Low) throughout the transfer.

The following figures show the TI synchronous single frame transfer and TI synchronous multiple frame transfer.

Figure 216 • TI Synchronous Serial Single Frame Transfer

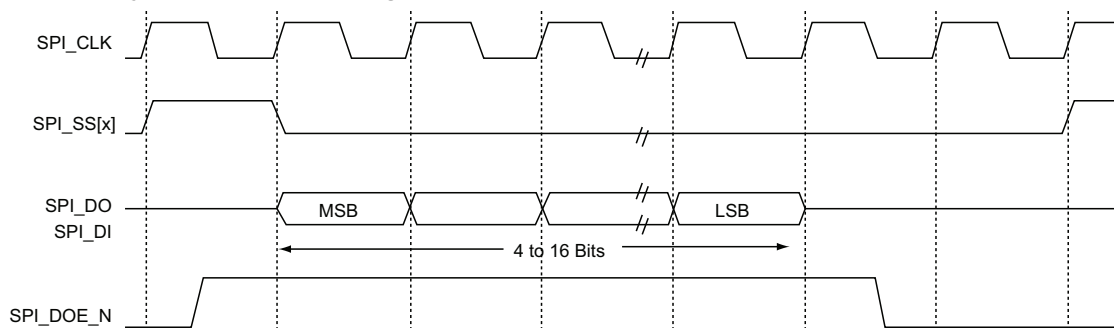
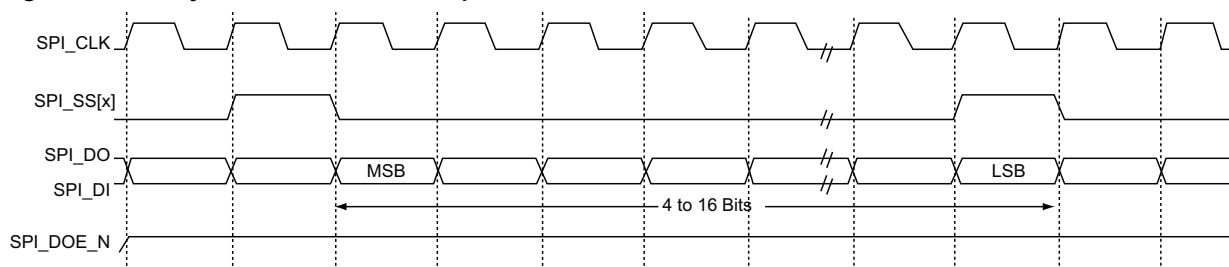


Figure 217 • TI Synchronous Serial Multiple Frame Transfer



14.2.2.5.1 TI Synchronous Serial Error Case Scenarios

When the SPI controller is configured for the TI synchronous serial protocol, while in slave mode, it responds to failure events. These failure events on slave select (SPI_X_SS[x]) and the slave clock (SPI_X_CLK) are described below:

- Withdrawal of SPI_X_CLK: In this case, the device pauses and resumes on reasserting the clock.
- Premature pulsing of slave select: If the slave select is pulsed during a data frame transmission, it will be ignored.
- Disconnecting the slave select before a transfer: The transfer is not initiated unless the pulse is issued.

14.2.2.6 Slave Protocol Engine

The slave protocol engine (SPE) implements a Microsemi-defined hardware protocol that allows the transfer of command and data from an SPI master to the SPI slave. The SPE controller logically sits between the SPI transmit/receive logic and the FIFOs. The SPE controller removes the command bytes and inserts status bytes from the data stream. Only one command byte is defined by Microsemi

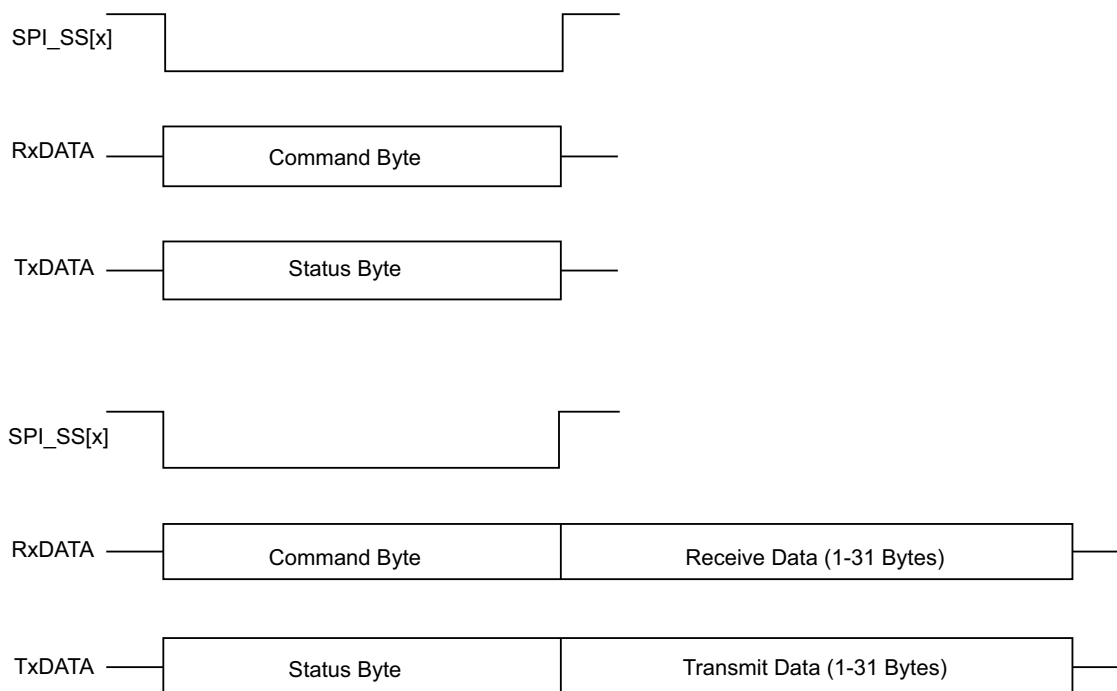
(POLL command). All other command bytes are user defined. To use the SPE, the [BIGFIFO](#), [AUTOSTATUS](#), [AUTOPOLL](#), [FRAMEURUN](#), and [SPS](#) bits should be set (refer to the SPI [CONTROL](#) register for bit definitions). The descriptions below assume that the frame size ([TXRXDF_SIZE](#)[[TXRXDFS](#)] field) is set to 8 bits, although other frame size settings are acceptable (up to 32 bits).

14.2.2.6.1 SPI Slave Frame Format

The frame format consists of a command frame followed by 0 to 31 data frames. The size of the command frame and data frame must be equal and is defined by [TXRXDFS](#). A typical use model would be to define the command frame as 8 bits followed by 31 bytes of data. This assumes [BIGFIFO](#) is set to 1 and [TXRXDFS](#) is set to 0x08.

The following figure shows the command and data bytes. Transmit and receive refer to the SPI peripheral as the slave. Data bytes are optional.

Figure 218 • SPE Command/Data Format



The first receive byte of the sequence after [SPI_X_SS\[x\]](#) asserts is always a command byte. The slave always responds with a status byte, which is the contents of the [HWSTATUS](#) register.

Note: Set two bits in the [HWSTATUS](#) register to facilitate additional handshaking schemes between SPI master and SPI slave.

14.2.2.6.2 POLL Command

All command bytes except the POLL command are stored in the receive FIFO. Once received, the CMD interrupt is generated. The command size can be set by the [CMD_SIZE](#) register and can be 1 to 32 bits wide, although typically commands and data will be 8 bits wide. The POLL command is encoded as 0xFF and is the only encoded command. All other command byte encodings are user defined. If a POLL command is received, the contents of the [HWSTATUS](#) register are sent back to the master and the POLL command is discarded. It will not be stored in the FIFO.

14.2.2.6.3 Hardware Status Frame

A hardware status frame is automatically sent back by the SPE in response to every command. It provides status information back to the master. The byte contains the contents of the [HWSTATUS](#) register.

14.2.2.6.4 Simple Commands

To send a command with no data to the slave, the master does the following:

1. Sends a POLL command and verifies that the slave is ready (no RXBUSY from [HWSTATUS](#) register).
2. This is repeated until the slave indicates it is ready.
3. The master sends the other command with no data. The command is queued in the receive FIFO for the slave to process.

14.2.2.6.5 Data Receive Operation

To send data to the slave, the master does the following:

1. Sends a POLL command and verifies that the slave is ready and can accept the data.
2. This is repeated until the slave indicates that it is ready and can accept the data (no RXBUSY from [HWSTATUS](#) register).
3. The master sends the write command and data bytes. On receiving, the slave stores the command and data bytes in the receive FIFO. After CMDSIZE bits have been received, the CMD interrupt is generated.
4. The hardware automatically set RxBUSY, if there are less than [PKTSIZE](#) storage locations left in the receive FIFO after the sequence completes.

Note: The slave reports under-run events having occurred, if no data is available for transmit.

14.2.2.6.6 Data Transmit Operation

To receive data from the slave, the master does the following:

1. Sends a POLL command and verifies that the slave is ready and can accept the data.
2. This is repeated until the slave indicates it is ready and can accept any associated command data (no RxBUSY).
3. The master sends a read command and any associated data bytes (for example, a read address). On receiving the sequence, the slave stores the command byte and data in the receive FIFO. User logic examines the command and data bytes and puts the requested data in the transmit FIFO. As soon as it has written [PKTSIZE](#) bytes to the transmit FIFO, the TxBUSY status bit in the [HWSTATUS](#) register will be cleared.
4. The master starts polling the device until the TxBUSY bit is cleared, indicating that the data is available.
5. The master now sends a read command followed by data words. The slave will return the contents of the [HWSTATUS](#) register and required data words.

14.2.2.6.7 Under-Run in Slave Mode

Under normal operating conditions, the SPI slave core in slave mode has a transmit FIFO under-run condition as the master initiates transfers when the slave transmit FIFO is empty (or attempts to transmit data faster than the slave processor loads data). The core's operation can be modified by setting [FRAMEURUN](#) ([CONTROL](#) register). Once set, the core will ignore the under-run conditions and simply transmit zero frames when the transmit FIFO is empty at the start of a series of frames. If the first data frame of a packet is read from the FIFO and transmitted, the under-run detection is enabled such that if the transmit FIFO fails to provide any of the rest of the data packet (assuming [SPI_X_SS\[x\]](#) is active for the whole packet), an over-run condition is signaled.

14.2.3 Initialization

This section describes the SPI initialization sequence, the SPI status at reset, and clock requirements. The SPI can be initialized by configuring the SPI [CONTROL](#) register and the [SOFT_RESET_CR](#) system registry.

14.2.3.1 Initialization Sequence

1. Select the type of transfer protocol by using the TRANSFPRTL bit of the SPI **CONTROL** register.
2. Enable SPI by writing 0 to the RESET bit of the **CONTROL** register.
3. Reset the transmit/receive buffers and the data frame size.

14.2.3.2 SPI Status at Reset

After SPI reset, the slave select (SPI_X_SS[X]) pins are held to the default values of logic High. After selecting SPI mode and enabling the SPI controller, the SPI_X_SS[x] lines are changed to the default values for each protocol. Refer to the [SPI Control Register \(CONTROL\)](#), page 528. After reset, the clock out (SPI_X_CLK) is at logic Low. At reset, the FIFOs are cleared and their respective read and write pointers are set to zero. Similarly, all the internal registers of the SPI controller are reset to their default values, as explained in the [SPI Register Summary](#), page 527.

An option is provided to reset the SPI peripherals by writing to bit 9 or bit 10 in the system register, SOFT_RESET_CR. The soft resets are encoded in the following table. At power-up, the reset signals are asserted 1. It keeps the SPI peripherals in a reset state. The SPI peripheral becomes active when the bit is set to 0, as shown in the table.

Table 496 • Soft Reset Bit Definitions for SPI Peripheral

Bit Number	Name	R/W	Reset Value	Description
10	SPI1_SOFTRESET	R/W	0x1	Controls reset input to SPI_1 0: Release SPI_1 from reset 1: Keep SPI_1 in reset
9	SPI0_SOFTRESET	R/W	0x1	Controls reset input to SPI_0 0: Release SPI_0 from reset 1: Keep SPI_0 in reset

14.2.3.3 SPI Clock Requirements

The SPI_0 and SPI_1 peripherals are clocked by APB_0_CLK on APB bus 0, and APB_1_CLK on APB bus 1. These clocks are derived from the main MSS clock, M3_CLK. Each APB clock can be programmed individually as M3_CLK is divided by 1, 2, 4, or 8. Refer to the [UG0449: SmartFusion2 and IGL002 Clocking Resources User Guide](#) for more information.

The SPI clocks in master mode are derived from APB_0_CLK / APB_1_CLK. Master mode and slave mode SPI data rates depend on the APB clock, as given below.

- Master mode SPI data rate
 - Programmable from APB_X_CLK/256 to APB_X_CLK/2
 - Programmable from APB_X_CLK /65536 to APB_X_CLK /256 in powers of 2
 - Maximum data rate is APB_X_CLK/2
- Slave mode SPI data rate operates up to
 - APB_X_CLK for frame sizes (frame size ≥ 8)
 - APB_X_CLK /2 for frame sizes (frame size 4 to 7)

14.2.4 Details of Operation

This section describes the SPI controller operation including FIFO, modes of data transfer, interrupts, and error handling.

14.2.4.1 SPI Transmit and Receive FIFO Flags

The SPI controller contains two, 4×32 (depth x width) FIFOs, as shown in [Figure 205 on page 505](#). One is for the receive side and the other is for the transmit side. The TXFIFOFUL and TXFIFOEMP bits of the **Status** register indicate the full or empty status of the transmit FIFO. The RXFIFOFUL and RXFIFOEMP bits of the **Status** register indicate the full or empty status of the receive FIFO. User logic can poll these bits to obtain the status of the corresponding FIFO.

For large data transfers, the full depth of transmit FIFO can be used by setting the number of data frames (more than one) in a burst (maximum is 64 k frames). When the interrupts are enabled, the TXDONE bit of the [RIS](#) register is asserted after all the data frames in the burst are sent.

For example, if the data frame size is set to 32 and the count is set to 2, the interrupt TXDONE is generated after every 2 words (each word is 32 bits). The default value for the frame count is one. The TXUNDERRUN and RXOVERFLOW bits of the [Status](#) register indicate that a FIFO under-run or FIFO overflow has occurred.

14.2.4.1.1 FIFO Under-Run Condition

If the transmit FIFO is accessed to transfer the data and there is no data in the FIFO, a transmit under-run error (TXUNDERRUN) is generated. This can be conditionally used to generate an interrupt. In this case, the transmission is assumed to have been lost and the application must catch the error and restart the transmission from the beginning. Internally, the transmit logic returns to an idle state and the entire transmission is deemed lost.

14.2.4.1.2 FIFO Overflow Condition

If the channel attempts to write into a receive FIFO which is already full, a receive overflow error (RXOVERFLOW) is generated. This can be conditionally used to generate an interrupt. In this case, the transmission continues but the data is now corrupted because the data frame is missing. It is assumed that the software clears the interrupt and recover; possibly by reading from the receive FIFO to clear the source of the interrupt, allowing more data to be received, or even by halting the transmission and resetting the SPI controller.

14.2.4.2 SPI Controller Modes of Data Transfer

There are two basic modes of transfer.

- Processor controlled mode: The data transfers are controlled by a firmware that either polls the [Status](#) register or responds to interrupts.
- PDMA controlled mode: The data transfers are autonomously controlled by the PDMA engine.

14.2.4.2.1 Processor Controlled Mode

In this mode, the size of the data frames (set in register [TXRXDF_SIZE](#)) and their numbers (set in the [CONTROL\[TXRXDFCOUNT\]](#) field) are specified. The data frame size specifies the number of bits being shifted out or being received per-frame. On completing each transfer, after a specified number of data frames (1 by default) are sent, an optional interrupt is generated. The SPI controller keeps track of the number of data frames so that special signals, like output enable, can be deactivated at the end of a transfer.

For example, to transmit one 17-bit word, the data frame size is set to 17, and the number of data frames is set to 1. Then depending on the operating mode, the 17 bits are transferred and the TXDATSENT [Status](#) register bit (0) is set. If enabled, an interrupt is also generated.

For example, consider the transmission of 64 KB of data to an external EEPROM from the processor controlled SPI controller. The data frame size is set to eight and the number of data frames per-transfer is set to one. After each transfer, the software must respond to the interrupt-transmit done-and-reload the FIFO until the 64 KB of data is sent. To improve throughput, the number of data frames per each transfer can be set to 4, in order to utilize the full depth of the transmit FIFO.

14.2.4.2.2 PDMA Controlled Mode

In PDMA mode, the interrupts are turned off and the PDMA controller uses SPI_X_TXRFM and SPI_X_RXAVAIL signals to govern the filling and emptying of the FIFOs. The SPI_X_RXAVAIL signal indicates that the data is available to be read and SPI_X_TXRFM indicates that the transmit is done and it is ready to receive more data.

For example, consider the transmission of 64 KB of data to an external EEPROM from a PDMA controlled SPI controller. The data frame size is set to eight and the number of data frames per-transfer is set to one. The transmit FIFO is repeatedly filled and emptied by the PDMA engine, using the SPI_X_TXRFM and SPI_X_RXAVAIL signals. In PDMA mode, the transmit done and receive data available interrupts are masked, and the PDMA engine is used to notify the application on completion.

14.2.4.3 Interrupts

Interrupts can be set up to signal the completion of a data frame transmission or reception. There is one interrupt signal from each SPI peripheral. The SPI_0_INT signal is generated by SPI_0 and is mapped to INTISR [2] in the Cortex-M3 processor nested vectored interrupt controller (NVIC). The SPI_1_INT signal is generated by SPI_1 and is mapped to INTISR [3] in the Cortex-M3 processor NVIC.

14.2.4.4 SPI Error Recovery and Handling

The SPI protocol defines only the packet formats for data transmission and does not include any error recovery strategy for physical layer protocols. Specifically, if an error occurs on a slave, such as failing to respond to the chip select or being overwhelmed with incoming data, the master will not necessarily be aware of it. The master and slave must therefore have prior knowledge of each other's capabilities before the transmission begins.

14.2.4.4.1 RX Overflow

An Rx overflow condition arises when the receive FIFO has not been emptied in time. As a result, the last write to the receive FIFO from the channel, overwrites the data that is received earlier and which is not read by the host processor. Eventually, the FIFO fills up and subsequent writes by the channel cause the Rx to overflow. The corrective action for the bus master is to read from the FIFO until the FIFO is empty. This can be checked by reading the FIFO status in the [Status](#) register.

14.2.4.4.2 TX Under-Run

A Tx under-run condition arises when a channel requests to send data while no data is available in the transmit FIFO. For example, when the SPI controller is operating in slave mode and receives a request to send data, when no data is available in transmit FIFO. The corrective action for the bus master is to write data into the transmit FIFO. The status flags (TXFIFOEMP or TXFIFOEMPXNT of the [Status](#) register) indicate whether the FIFO is empty or will be empty after the next read operation.

14.3 How to Use SPI

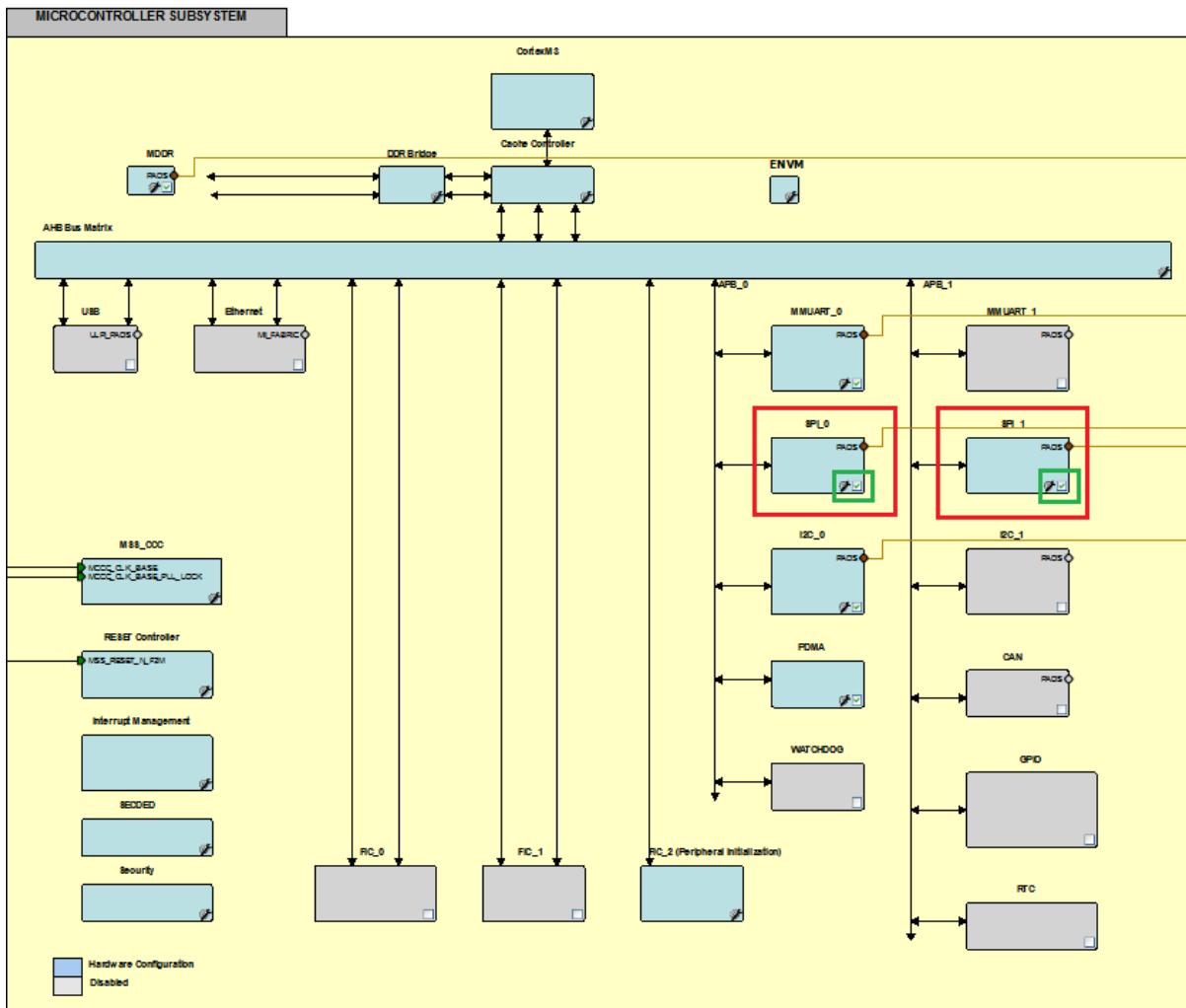
This section describes how to use SPI in an application.

14.3.1 Design Flow

The following steps are used to enable the SPI in the application by using Libero SoC.

1. Enable SPI_0 and/or SPI_1 instance by using the MSS configurator in the application, as shown in the following figure.

Figure 219 • Enable SPI



2. Configure the number of Slaves to 1 and the ports of enabled SPI_0 instance to an IO or Fabric by using MSS SPI_0 Configurator as shown in the following figures. Click the highlighted **Users Guide** button to find more information on SPI configuration details.

Figure 220 • MSS SPI Configurator - Connection Type IO

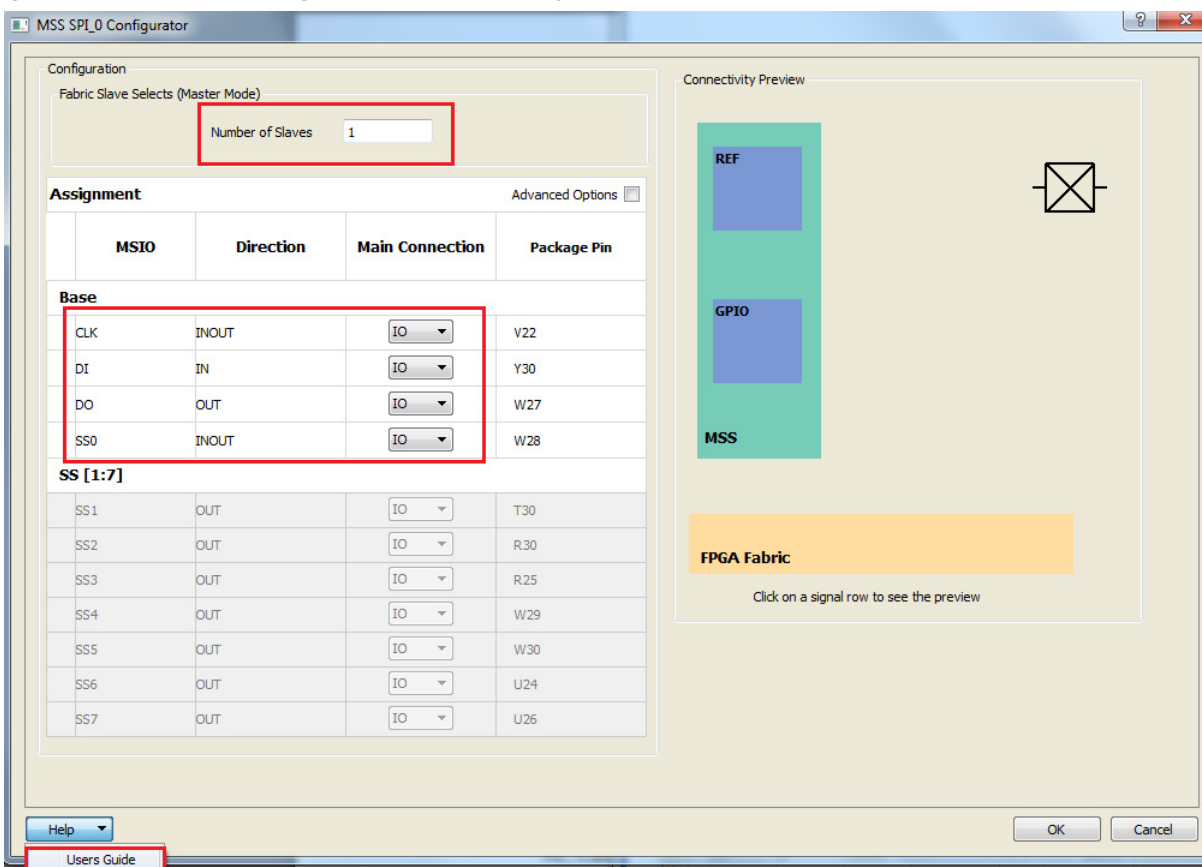
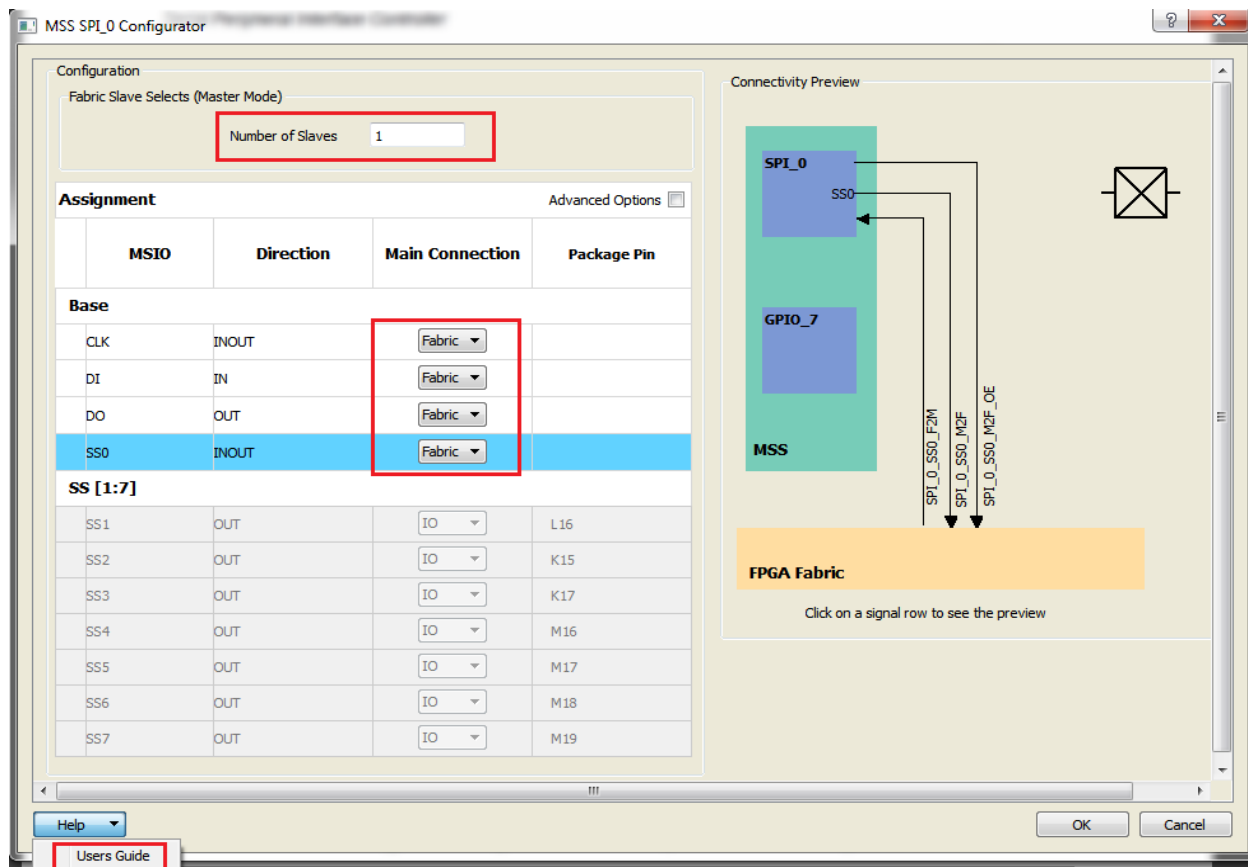


Figure 221 • MSS SPI Configurator - Connection Type Fabric

3. The SPI_0 interface signals in the MSS component are shown in the following figure.

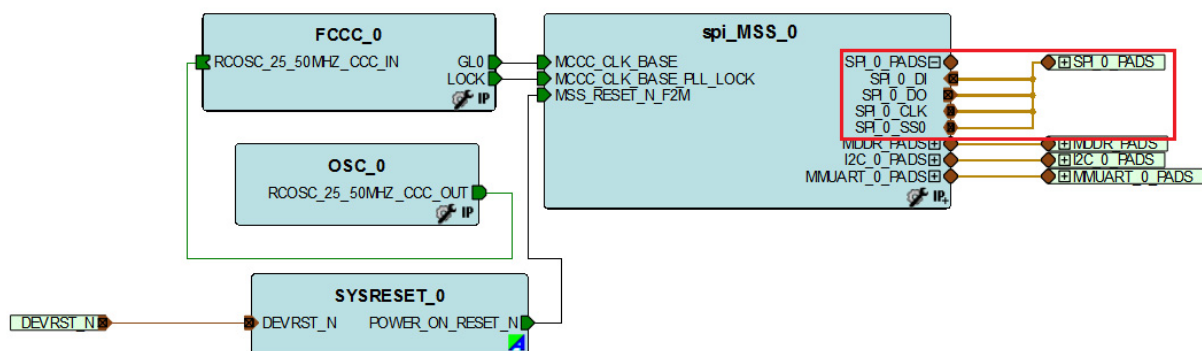
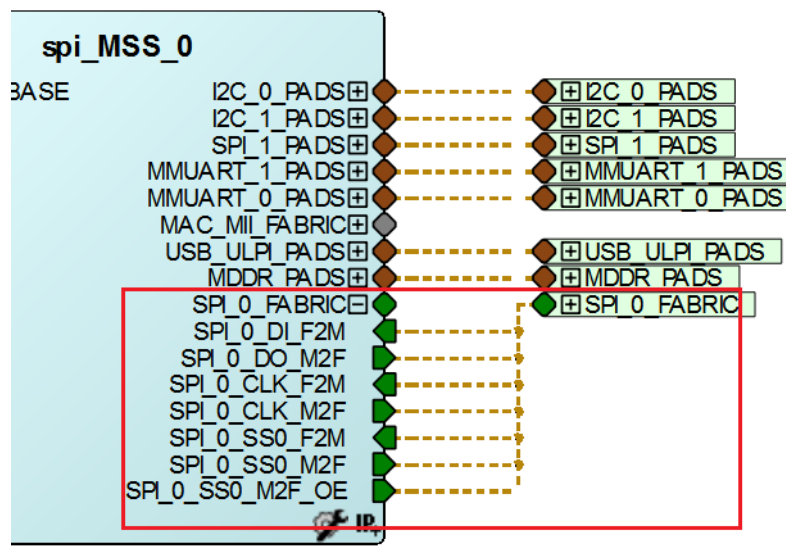
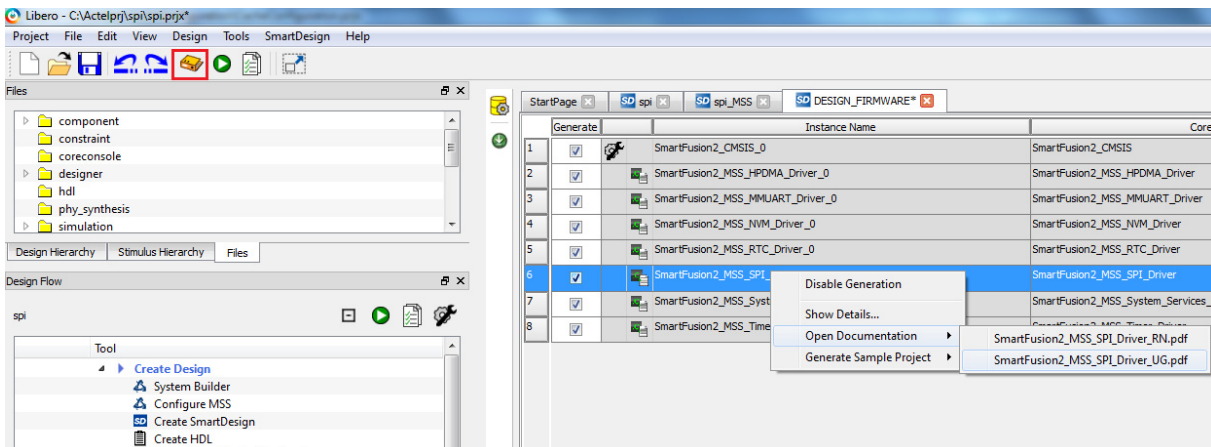
Figure 222 • SPI Interface Signals - Connection Type IO

Figure 223 • SPI Interface Signals - Connection Type Fabric



4. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component**.
 For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace are included into the project. Click the **Configure firmware** button highlighted in the following figure to find the SPI driver information.

Figure 224 • SPI Driver User's Guide



5. Click **Generate Bitstream** under **Program Design** to complete the .fdb file generation.
6. Double-click the **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_spi firmware driver. The firmware driver, mss_spi (mss_spi.c and mss_spi.h), which provides a set of functions for controlling the MSS SPIs can also be downloaded from the Microsemi Firmware Catalog.

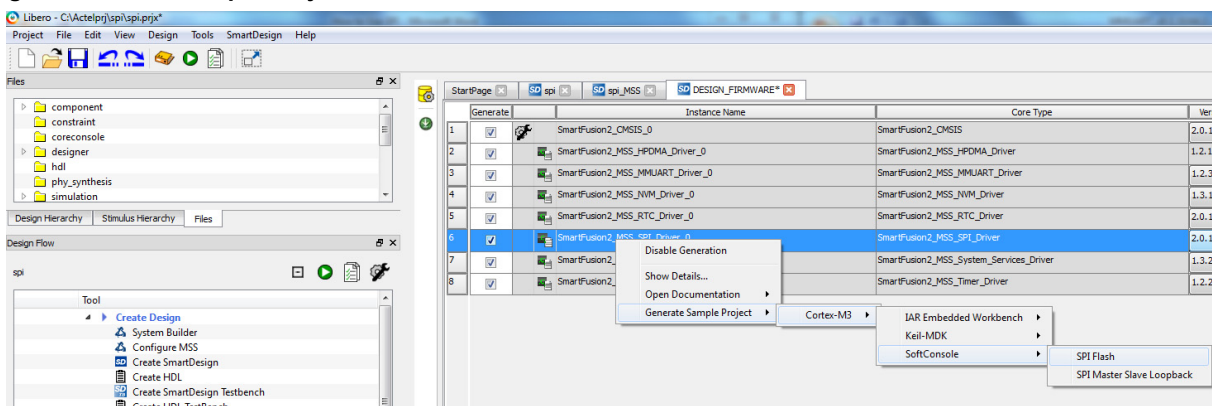
The following table lists main APIs for SPI. For complete information on the APIs, refer to the [SmartFusion2 MSS SPI Driver UG](#), as shown in the preceding figure.

Table 497 • MSS SPI APIs

Category	API	Description
Initialization and configuration functions	MSS_SPI_init()	Initializes the SPI
	MSS_SPI_configure_master_mode()	Configures the protocol mode, serial clock speed and frame size for a specific target SPI slave device
	MSS_SPI_configure_slave_mode()	Configures a MSS SPI block for operations as a SPI slave device
Data transfer functions	MSS_SPI_set_slave_select()	Selects a specific slave by a MSS SPI master
	MSS_SPI_transfer_frame()	Used by a MSS SPI master to transmit and receive a frame up to 32 bits long
	MSS_SPI_transfer_block()	Used by MSS SPI masters to transmit and receive blocks of data
	MSS_SPI_clear_slave_select()	To deselect a specific slave by a MSS SPI master
	MSS_SPI_set_slave_tx_frame()	Used by MSS SPI slaves to specify the frame that will be transmitted
	MSS_SPI_set_frame_rx_handler()	Used by MSS SPI slaves to specify the receive handler function
DMA block transfer functions	MSS_SPI_disable()	To temporarily disable a MSS SPI
	MSS_SPI_set_transfer_byte_count()	Specifies the number of bytes
	MSS_SPI_enable()	To re-enable a MSS SPI
	MSS_SPI_tx_done()	To find out if a DMA controlled transfer has completed

7. For more information on SPI usage, the sample projects are available and can be generated, as shown in the following figure.

Figure 225 • SPI Sample Project



Note: The MSS SPI does not support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for more information.

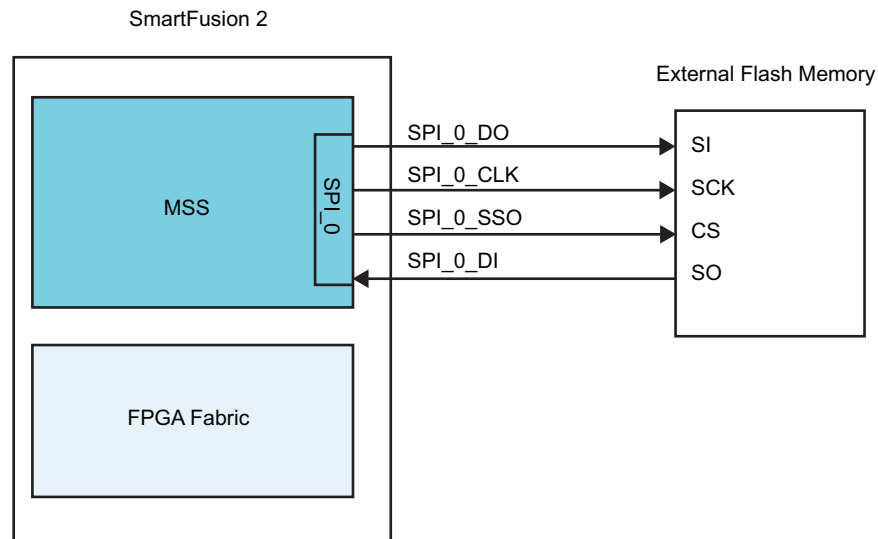
14.3.2 SPI Use Models

14.3.2.1 Use Model 1: Accessing the External SPI Flash Using MSS SPI_0

The external SPI flash can be interfaced to either the MSS SPI_0 or SPI_1 peripherals of the SmartFusion2 MSS. In this example, the external SPI flash is interfaced to MSS SPI_0. The MSS SPI_0 is configured as a master with the slave select line connected to the chip select of the external SPI Flash.

The following figure shows interfacing the external SPI flash to MSS SPI_0.

Figure 226 • Interfacing External SPI Flash to MSS SPI_0-Block Diagram

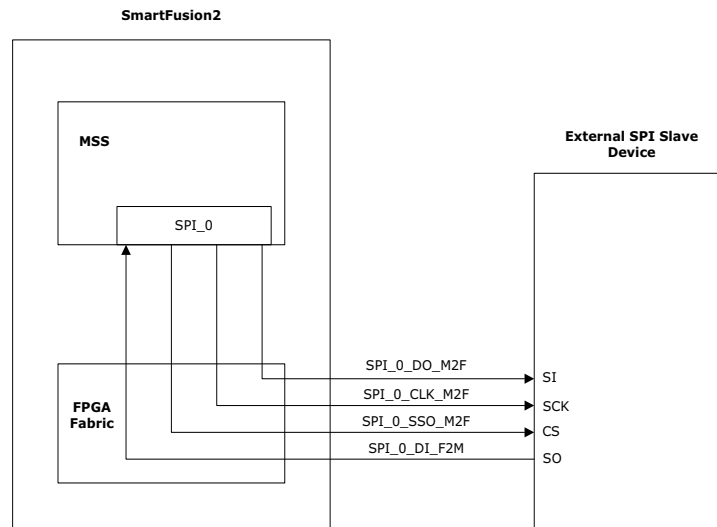


Follow the instructions provided in the [Design Flow](#), page 520 to configure SPI_0 in the application. The external SPI flash is the target SPI slave device in this example.

14.3.2.2 Use Model 2: Interfacing Any SPI Slave Device Using MSS SPI Routing Through Fabric

The external SPI slave devices like serial display device can be interfaced using SmartFusion2 MSS SPI routing through fabric ports. The fabric ports can be brought out to the GPIO header to interface the SPI slave device with SmartFusion2 device. Follow the instructions provided in the [Design Flow](#), page 520 to configure SPI_0 with connection type fabric in the application. Double click **IO- Constraints** under **Edit Constraints** in the Libero SoC design flow window to configure the appropriate GPIO header Pin numbers for MSS SPI_0 fabric ports to brought out the SPI signals to interface the SPI slave device with SmartFusion2 Device.

Figure 227 • Interfacing External SPI Slave Device Using MSS SPI Routing Through Fabric Block Diagram



14.3.2.2.1 Software Design Flow

Initialization of the SPI Peripheral

Initialize the SPI instance SPI_0 by using MSS_SPI_init API.

SPI Master Mode Configuration

Configure MSS SPI_0 in master mode by using MSS_SPI_configure_master_mode API. The SPI peripheral generates a serial clock and data to the slave device.

The following parameters are required to configure SPI_0 so it will operate as a SPI Master.

- SPI_0 instance data structure
- Target SPI slave (SPI flash, for example)
- Serial peripheral interface operating mode
- Divider value to generate serial interface clock signal from PCLK
- Frame bit length

Data Transfer

Follow the below steps to transfer data to SPI flash or to receive data from SPI flash.

- Set the slave by using MSS_SPI_set_slave_select API.
- Transfer the data frame by using MSS_SPI_transfer_frame API.
- Deselect the slave after data transfer by using MSS_SPI_clear_slave_select API.

Refer to the [TU0548: Accessing Serial Flash Memory Using SPI Interface - Libero SoC v11.6 SoC and Keil uVision Flow for SmartFusion2 Tutorial](#) for more information on how to access Flash memory through SPI interface.

Refer to the **In-Application Programming** chapter of *UG0451: IGLOO2 and SmartFusion2 Programming User Guide* for more information on usage of SPI_0 peripheral in IAP.

14.4 SPI Register Map

This section provides SPI registers along with the address offset, functionality, and bit definitions.

14.4.1 SYSREG Configuration Register Summary

The registers listed in the following table control the behavior of the SPI peripherals. Refer to the [System Register Block](#), page 670 for a detailed description of each register and bit.

Table 498 • SYSREG Control Registers

Register Name	Register Type	Flash Writer Protect	Reset Source	Description
SOFT_RESET_CR	RW-P	Bit	SYSRESET_N	Soft reset control
LOOPBACK_CR	RW-P	Register	SYSRESET_N	Loop back control
PERIPH_CLK_MUX_SEL_CR	RW-P	Register	PORESET_N	Peripheral clock MUX select

14.4.2 SPI Register Summary

The following table summarizes each of the SPI registers described in this document. The SPI_0 base address resides at 0x40001000 and extends to address 0x40001FFF in the Cortex-M3 processor memory map. The SPI_1 base address resides at 0x40011000 and extends to address 0x40011FFF in the Cortex-M3 processor memory map.

Table 499 • SPI Register Summary

Register Name	Address Offset	R/W	Reset Value	Description
CONTROL	0x00	R/W	0x80000102	Control register
TXRXDF_SIZE	0x04	R/W	0x04	Transmit and receive data frame size
Status	0x08	R	0x2440	Status register
INT_CLEAR	0x0C	W	0x0	Interrupt clear register
RX_DATA	0x10	R	0x0	Receive data register
TX_DATA	0x14	W	0x0	Transmit data register
CLK_GEN	0x18	R/W	0x07	Output clock generator (master mode)
SLAVE_SELECT	0x1C	R/W	0x0	Specifies slave selected (master mode)
MIS	0x20	R	0x0	Masked interrupt status
RIS	0x24	R	0x0	Raw interrupt status
CONTROL2	0x28	R/W	0x0	Control bits for enhanced modes
COMMAND	0x2C	R/W	0x0	Command register
PKTSIZE	0x30	R/W	0x0	Packet size
CMD_SIZE	0x34	R/W	0x0	Command size
HWSTATUS	0x38	R/W	0x0	Slave hardware status
STAT8	0x3C	R	0x44	Status register
CTRL0	0x40	R/W	0x02	Aliased CONTROL register bits 7:0. This register allows byte operations from an 8-bit processor in the fabric. It is not intended for access from internal MSS masters.

Table 499 • SPI Register Summary (continued)

Register Name	Address Offset	R/W	Reset Value	Description
CTRL1	0x44	R/W	0x01	Aliased CONTROL register bits 15:8. This register allows byte operations from an 8-bit processor in the fabric. It is not intended for access from internal MSS masters.
CTRL2	0x48	R/W	0x0	Aliased CONTROL register bits 23:16. This register allows byte operations from an 8-bit processor in the fabric. It is not intended for access from internal MSS masters.
CTRL3	0x4C	R/W	0x0	Aliased CONTROL register bits 25:24. This register allows byte operations from an 8-bit processor in the fabric. It is not intended for access from internal MSS masters.

14.4.3 SPI Register Details

This section describes the SPI registers in detail.

14.4.3.1 SPI Control Register (CONTROL)

The following table provides details about the SPI Control register. Using this register, the SPI mode (Master/Slave), the type of the protocol it uses, and the data frame count can be set.

Table 500 • CONTROL

Bit Number	Name	R/W	Reset Value	Description
31	RESET	R/W	1	0: SPI is enabled 1: SPI is held in Power reset state.
30	OENOFF	R/W	0	0: SPI output enable active as required 1: SPI output enable is not asserted. Allows multiple slaves to share a single slave select signal with a single master.
29	BIGFIFO	R/W	0	Alters FIFO depth when frame size is [4-8] bits. 0: FIFO depth is 4 frames. 1: FIFO depth is 32 frames when frame size is [9-16] bits FIFO depth is 16; and when frame size is [17-32] bits FIFO depth is 8.
28	CLKMODE	R/W	0	Specifies the methodology used to calculate the SPICLK divider. 0: $SPICLK = 1 / (2^{CLK_GEN + 1})$ where $CLK_GEN = 0$ to 15. 1: $SPICLK = 1 / (2 \times (CLK_GEN + 1))$ where $CLK_GEN = 0$ to 255.
27	FRAMEURUN	R/W	0	0: The under-runs are generated whenever a read is attempted from an empty transmit FIFO. 1: The under-run condition will be ignored for the complete frame, if the first data frame read resulted in a potential overflow; that is, the slave was not ready to transmit any data. If the first data frame is read from the FIFO and transmitted, an under-run will be generated, when the FIFO becomes empty for any of the remaining packet frames (that is, while SSEL is active). Master operation does not create a transmit FIFO under-run condition.
26	SPS	R/W	0	Defines slave select behavior. Refer to Table 494 , page 507.
25	SPH	R/W	0	Clock phase
24	SPO	R/W	0	Clock polarity
[23:8]	TXRXDFCOUNT	R/W	0001	Number of data frames to be sent or received. Counts from 1. Maximum value is 0xFFFF.

Table 500 • CONTROL (continued)

Bit Number	Name	R/W	Reset Value	Description
7	INTTXTURUN	R/W	0	Interrupt on transmit the under-run 0: Interrupt disabled 1: Interrupt enabled
6	INTRXOVRFLO	R/W	0	Interrupt on receive overflow 0: Interrupt disable 1: Interrupt enabled
5	INTTXDATA	R/W	0	Interrupt on transmit data 0: Interrupt disabled 1: Interrupt enabled
4	INTRXDATA	R/W	0	Interrupt on receive data 0: Interrupt disabled 1: Interrupt enabled
[3:2]	TRANSFPRTL	R/W	0	Transfer protocol Decode: 0b00: Motorola SPI 0b01: TI synchronous serial 0b10: National Semiconductor MICROWIRE 0b11: Reserved The transfer protocol cannot be changed while the SPI is enabled.
1	MODE	R/W	1	SPI implementation 0: Slave 1: Master (default)
0	ENABLE	R/W	0	Core enable 0: Disable (default) 1: Enable The core will not respond to external signals (SPI_X_DI, SPI_X_DO) until this bit is enabled. SPI_X_CLK is driven low and SPI_X_DOE_N and SPI_X_SS (slave select) are driven inactive.

14.4.3.2 SPI TxRx Data Frame Register (TXRXDF_SIZE)

The following table provides details about the Transmit Receive Data Frame register. The width of the data frame is set using this register.

Table 501 • TXRXDF_SIZE

Bit Number	Name	R/W	Reset Value	Description
[31:6]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
[5:0]	TXRXDFS	R/W	0x04	Transmit and receive data size. Maximum value is 32. Number of bits shifted out and received per frame (count starts from 1). In National Semiconductor MICROWIRE mode, this is the number of shifts to be done after the control byte is sent. This register must be set before SPI is enabled. Writes to this register are ignored after the SPI is enabled.

14.4.3.3 SPI Status Register (STATUS)

The following table provides the SPI Status register details. This register indicates the state of SPI such as Tx/Rx FIFO, Tx under-run, and Rx overflow.

Table 502 • Status

Bit Number	Name	R/W	Reset Value	Description
[31:15]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
14	ACTIVE	R		SPI is still transmitting or receiving data.
13	SSEL	R		Current state of SPI_X_SS[0]
12	FRAMESTART			0: SPI output enable is active as required. 1: SPI output enable is not asserted. Allows multiple slaves to share a single slave select signal with a single master.
11	TXFIFOEMP NXT	R	0	Transmit FIFO empty on next read
10	TXFIFOEMP	R	1	Transmit FIFO is empty
9	TXFIFOFUL NXT	R	0	Transmit FIFO full on next write
8	TXFIFOFUL	R	0	Transmit FIFO is full
7	RXFIFOEMP NXT	R	0	Receive FIFO empty on next read
6	RXFIFOEMP	R	1	Receive FIFO empty
5	RXFIFOFUL NXT	R	0	Receive FIFO full on next write
4	RXFIFOFUL	R	0	Receive FIFO is full
3	TXUNDERRUN	RO	0	No data available for transmission. The channel cannot read data from the transmit FIFO because the transmit FIFO is empty. Certainly this can only be raised in slave mode because the master will not attempt to transmit unless there is data in FIFO.
2	RXOVERFLOW	RO	0	Channel is unable to write to receive FIFO as it is full. Applies to master and slave modes.
1	RXDATRCED	RO	0	When set, it indicates that the number of frames specified by TXRXDFCOUNT has been received and can be read. Applies to master and slave modes.
0	TXDATSENT	RO	0	When set, it indicates that the numbers of frames specified by TXRXDFCOUNT has been sent. Applies to master and slave modes.

Notes:

- Bits [11:4] correspond to FIFO status.
- None of these status bits are sticky. During run-time, the status of these bits reflects the current status of SPI.
- To determine the cause of an interrupt, the masked interrupt status (MIS) register must be read.

14.4.3.4 SPI Interrupt Clear Register (INT_CLEAR)

The following table describes the Interrupt Clear register. A read to this register has no effect. It returns all zeroes.

Table 503 • INT_CLEAR

Bit Number	Name	R/W	Reset Value	Description
[31:6]	Reserved	W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
5	SSEND	W		Write one to clear the interrupt
4	CMDINT	W		Write one to clear the interrupt
3	TXCHUNDRUN	W	0	Transmit channel under-run
2	RXCHOVRFLW	W	0	Receive channel over flow
1	RXRDYCLR	W	0	Clears receive ready (RX_RDY)
0	TXDONECLR	W	0	Clears transmit done (TX_DONE)

14.4.3.5 SPI Receive Data Register (RX_DATA)

The following table describes the Receive Data register.

Table 504 • RX_DATA

Bit Number	Name	R/W	Reset Value	Description
[31:0]	RXDATA	R	0	Received data. Reading this clears the register of the received data.

14.4.3.6 SPI Transmit Data Register (TX_DATA)

The following table describes the Transmit Data register.

Table 505 • TX_DATA

Bit Number	Name	R/W	Reset Value	Description
[31:0]	TXDATA	W	0	Data to be transmitted. Writing to this clears the last data transmitted.

14.4.3.7 SPI SCLK Generation Register (CLK_GEN)

The following table describes the clock modes used to calculate the SPICLK divider. [Table 507](#), page 532 describes the SPICLK rates in different modes.

Table 506 • CLK_GEN

Bit Number	Name	R/W	Reset Value	Description
[31:8]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

Table 506 • CLK_GEN (continued)

[7:0]	CLK_GEN	R/W	0	Specifies the methodology used to calculate the SPICLK divider. CLK_MODE = 0: $SPICLK = 1 / (2^{CLK_GEN + 1})$ where CLK_GEN = 0 to 15. CLK_MODE = 1: $SPICLK = 1 / (2 \times (CLK_GEN + 1))$ where CLK_GEN = 0 to 255.
-------	---------	-----	---	---

Table 507 • CLK_MODE Example, APB Clock = 153.8 MHz

CLK_MODE=0		CLK_MODE=1	
$SPICLK = 1 / (2^{CLKRATE + 1})$ where CLKRATE = 0 to 15		$SPICLK = 1 / (2 \times (CLKRATE + 1))$ where CLKRATE = 0 to 255	
CLKRATE	SPI Clock	CLKRATE	SPI Clock
0	76,900,000	0	76,900,000
1	38,450,000	1	38,450,000
2	19,225,000	2	25,633,333.33
3	9,612,500	3	19,225,000
4	4,806,250	4	15,380,000
5	2,403,125	5	12,816,666.67
6	1,201,562.5	6	10,985,714.29
7	600,781.25	7	9,612,500
8	300,390.625	8	8,544,444.444
9	150,195.312	9	7,690,000
10	75,097.656	10	6,990,909.091
11	37,548.828	11	6,408,333.333
12	18,774.414	12	5,915,384.615
13	9,387.207	13	5,492,857.143
14	4,693.603	14	5,126,666.667
15	2,346.801	15	4,806,250
		⋮	⋮
		255	300,390.625

14.4.3.8 SPI Slave Select Register

The following table describes the register that specifies the slave that has been selected.

Table 508 • SLAVE_SELECT

Bit Number	Name	R/W	Reset Value	Description
[31:8]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

Table 508 • SLAVE_SELECT (continued)

7:0	SLAVE SELECT	R/W	0	Specifies the slave selected. Writing one to a bit position selects the corresponding slave. SLAVESELECT[7:1] are available at the FPGA fabric interface, while SLAVESELECT[0] is available at the SPI_X_SS[0] pin. The slave select output polarity is active low. In TI mode the slave select output is inverted to become active high.
-----	--------------	-----	---	---

14.4.3.9 SPI Masked Interrupt Status Register

The following table describes the Masked Interrupt Status (MIS) register. It is a read-only register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.

Table 509 • MIS

Bit Number	Name	R/W	Reset Value	Description
[31:6]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
5	SSEND	R		Indicates that SPI_X_SS[x] has gone inactive. When this is high, the interrupt is active.
4	CMDINT	R		Indicates that the number of frames set by the CMDSIZE register has been received as a single packet of frames (SPI_X_SS[x] held active). When this is high, the interrupt is active.
3	TXCHUNDDMSKINT	R	0	Masked interrupt status. Reading this returns interrupt status. Masked interrupt status = Raw interrupt status and interrupt mask (CONTROL register). MIS = RIS and CONTROL[7:4]. Masked status of transmit channel under-run TXCHUNDDMSKINT = TXCHUNDRINT and INTTXUNRRUN
2	RXCHOVRFMSKINT	R	0	Masked status of receive channel overflow. RXCHOVRFMSKINT = RXCHOVRFINT and INTRXOVRFLO
1	RXRDYMSKINT	R	0	Masked status of receive data ready (data received in FIFO). RXRDYMSKINT = RXRDY and INTTXDATA
0	TXDONEMSKINT	R	0	Masked status of transmit done (data shifted out) TXDONEMSKINT = TXDONE and INTRXDATA

14.4.3.10 SPI Raw Interrupt Status Register

The following table describes the Raw Interrupt Status (RIS) register. This register returns the current raw status value, prior to masking, of the corresponding interrupt.

Table 510 • RIS

Bit Number	Name	R/W	Reset Value	Description
[31:6]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
5	SSEND	R/W		Indicates that SPI_X_SS[x] has gone inactive.

Table 510 • RIS

Bit Number	Name	R/W	Reset Value	Description
4	CMDINT	R/W		Indicates that the number of frames set by the CMDSIZE register has been received as a single packet of frames (SPI_X_SS[x] held active).
3	TXCHUNDR	R	0	RAW interrupt status. Reading this returns raw interrupt status. Raw status of transmit channel under-run
2	RXCHOVRF	R	0	Raw status of receive channel overflow
1	RXRDY	R	0	Receive data ready (data received in FIFO)
0	TXDONE	R	0	Raw status of transmit done (data shifted out)

14.4.3.11 SPI Control2 Register

The following table describes the Control2 register details as the terminal frame counter, SPI slave select, and auto status of SPI.

Table 511 • CONTROL2

Bit Number	Name	R/W	Reset Value	Description
[31:6]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
5	INTEN_SSEND	R/W		Indicates that SPI_X_SS[x] has gone inactive.
4	INTEN_CMD	R/W		Indicates that the number of frames set by the CMDSIZE register have been received as a single packet of frames (SPI_X_SS[x] held active).
3	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
2	DISFRMCNT	R/W	0	0: The internal frame counter is active. When the counter reaches the programmed limit, it will pause the current SPI transfer inserting idle cycles and generate the appropriate interrupts. 1: The internal frame counter is not active. The core transmits data until the transmit FIFO empties. The FRAMECNT (CONTROL register) should also be programmed to zero.
1	AUTOPOLL	R/W	0	0: No effect 1: The first receive frame after SPI_X_SS[0] is active. It is discarded (not written to the FIFO) and supports the POLL function.
0	AUTOSTATUS	R/W	0	0: No effect 1: The first transmitted frame (slave mode) contains the hardware status, not data from the transmit FIFO.

14.4.3.12 SPI Command Register

The following table describes the Command register..

Table 512 • COMMAND

Bit Number	Name	R/W	Reset Value	Description
[31:7]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
6	TXNOW	R/W	0	0: No effect 1: Writing one clears the TxBUSY bit in slave mode immediately rather than waiting for PKTSIZE frames to be available, telling the master that there is data available. This is intended to use when less than the programmed PKTSIZE data frames are being transmitted, removing the requirement to transmit PKTSIZE frames. This bit stays set until the first data frame is transmitted.
5	AUTOSTALL	R/W		0: No effect 1: Writing one will cause the master to delay transmission until the transmit FIFO contains the number of frames specified by the PKTSIZE register. This guarantee that the frames are transmitted with no idle cycles or time gaps between them. This bit will be automatically cleared as soon as the core starts transmitting the frames.
4	CLRFRAMECNT	R/W		0: No effect 1: Writing one clears the internal frame counter. This bit always reads as zero. The counter is also cleared when the core is disabled, CTL1, or CTL2 are written (that is, the frame count limit changed).
3	TXFIFORST	R/W	0	0: No effect 1: Writing one resets the Tx FIFO. This bit always reads as zero.
2	RXFIFORST	R/W	0	0: No effect Writing one resets the Rx FIFO. This bit always reads as zero.
1	AUTOEMPTY	R/W	0	0: No effect 1: Writing one causes the SPI core to automatically discard any further received data until the number of frames requested in the FRAMECNT register has been received or (in slave mode) SSEL goes inactive. This bit will stay set until all the frames are complete or it is cleared.
0	AUTOFILL	R/W	0	0: No effect 1: Writing one causes the SPI core to automatically fill the transmit FIFO with zeros to match the number of frames requested in the FRAMECNT register. Typically, the five command bytes must be written to the TxDATA register and then this bit must be set. Data can be read from the receive FIFO until the complete set of frames has been read. This bit will stay set until all the frames are complete or it is cleared.

14.4.3.13 SPI Packet Size Register

The following table provides the details of the Packet Size registers that are used to set the SPI CMD/data frame size.

Table 513 • PKTSIZE

Bit Number	Name	R/W	Reset Value	Description
[31:8]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
[7:0]	PKTSIZE	R/W	0	Sets the size of the SPI CMD/data frame. PKTSIZE cannot be greater than the FIFO size.

14.4.3.14 SPI Command Size Register

The following table describes the Command size register.

Table 514 • CMD_SIZE

Bit Number	Name	R/W	Reset Value	Description
[31:8]	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
[7:0]	CMDSIZE	R/W	0	Number of frames after SPI_SS[0] going active that the CMD interrupt should be generated. This controls the RxCMD interrupt. The internal counters count frames from SPI_SS[0] going low. It automatically resets and starts counting again once SSEL goes inactive. In TI mode, back-to-back frames are counted, any gaps in data causes the counter to start counting again.

14.4.3.15 SPI Hardware Status Register

The following table describes the Hardware Status register. This register allows the Cortex-M3 processor to control the hardware Status register used in the slave protocol controller.

Table 515 • HWSTATUS

Bit Number	Name	R/W	Reset Value	Description
[31:4]	Not used	R/W	0	These bits are undefined. The value that the slave transmits depends on the data that is queued in the transmit FIFO.
[3:2]	USER	R/W	0	These bits are set by the CPU. Their function is undefined but could be used to send additional status or request information to the master.
1	TXBUSY	R/W	0	0: Master may request the requested data. There are PKTSIZE frames of data in the transmit FIFO (when AUTOPOLL is set to PKTSIZE - 1) 1: Indicates not ready to transmit data.
0	RXBUSY	R/W	0	1: Indicates that the receive buffer is busy (not empty). 0: Indicates that up to PKTSIZE frames of command followed by data may be sent to the slave.

14.4.3.16 SPI Status 8 Register

The following table describes the SPI status 8 (STAT8) register. This register allows the important status bits to be read as a single 8-bit value. This reduces the overhead of checking the Status register bits when an 8-bit processor is being used.

Table 516 • STAT8

Bit Number	Equivalent STATUS Register Bit Position	Name	R/W	Reset Value	Description
[31:8]		Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7	14	ACTIVE	R	0	SPI is still transmitting the data
6	13	SSEL	R	0	Current state of SPI_X_SS[0]
5	3	TXUNDERRUN	R	0	Transmit FIFO underflowed
4	2	RXOVERFLOW	R	0	Receive FIFO overflowed
3	8	TXFIFOFUL	R	0	Transmit FIFO is full
2	6	RXFIFOEMP	R	0	Receive FIFO is empty
1	0 and 1	DONE	R	0	The number of request frames have been transmitted and received.
0	12	FRAMESTART	R	0	Next frame in receive FIFO was received after SPI_X_SS[x] went active (command frame).

15 Inter-Integrated Circuit Peripherals

Philips inter-integrated circuit (I²C) is a two wire serial bus interface that provides data transfer between many devices. SmartFusion2 SoC FPGAs contain two identical I²C peripherals in the microcontroller subsystem (MSS I²C_0 and MSS I²C_1), that provide a mechanism for serial communication between the SmartFusion2 device and external I²C compliant devices.

SmartFusion2 I²C peripherals support the following protocols:

- I²C protocol as per v2.1 specification
- SMBus protocol as per v2.0 specification
- PMBus protocol as per v1.1 specification

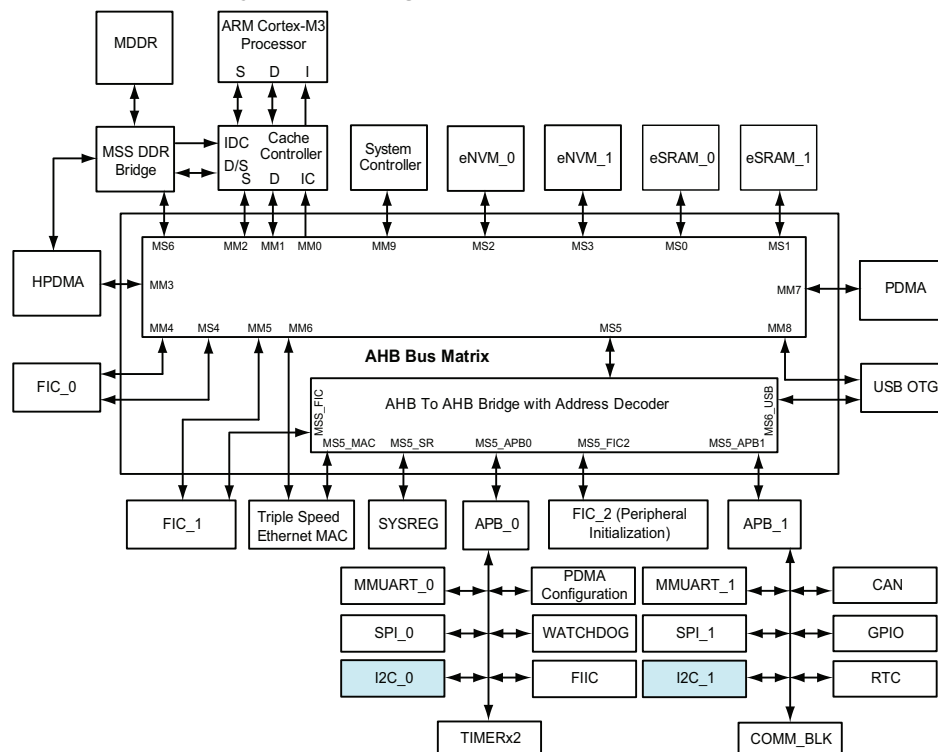
15.1 Features

SmartFusion2 I²C peripherals support the following features:

- Master and Slave modes
- 7-bit addressing format and data transfers up to 100 Kbit/s in Standard mode and up to 400 Kbit/s in Fast mode
- Multi-master collision detection and arbitration
- Own slave address and general call address detection
- Second slave address detection
- System management bus (SMBus) timeout and real-time idle condition counters
- Optional SMBus signals, SMBSUS_N and SMBALERT_N, which are controlled through the APB interface
- Input glitch or spike filters

The following figure shows the I²C peripherals within the MSS. The I²C peripherals are connected to the advanced high-performance bus (AHB) Matrix through the advanced peripheral bus (APB) interfaces (APB_0 and APB_1).

Figure 228 • Microcontroller Subsystem Showing I²C Peripherals



15.2 Functional Description

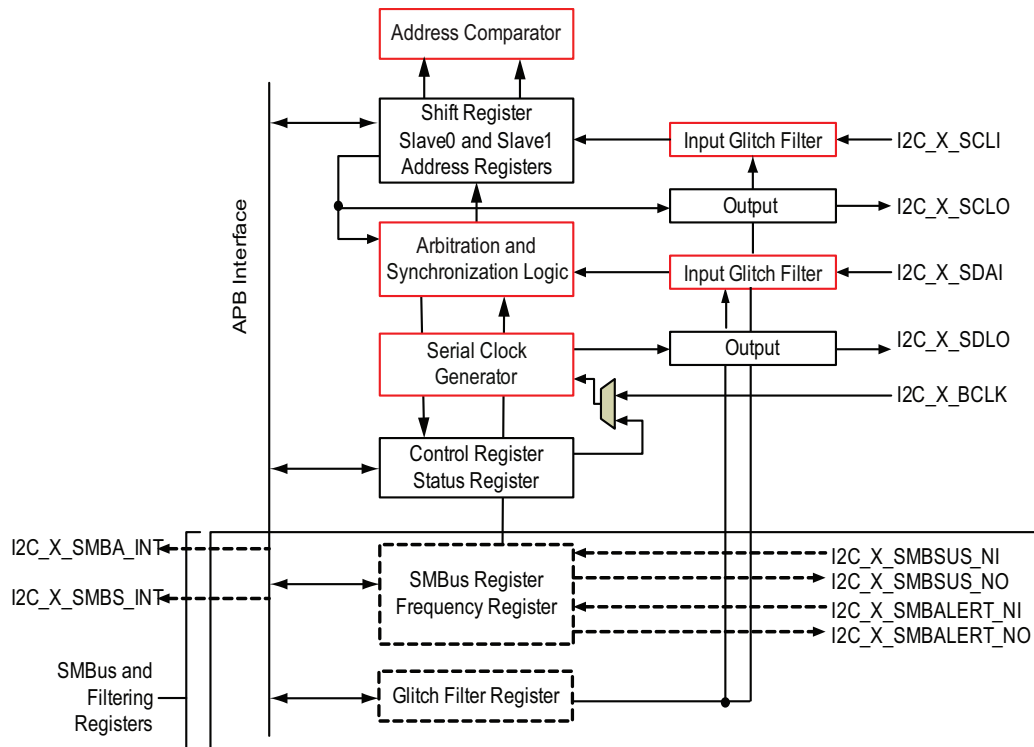
This section provides a detailed description of the I²C peripherals.

15.2.1 Architecture Overview

The I²C peripherals consist mainly of the following components (shown in the following figure).

- Input Glitch Filter
- Arbitration and Synchronization Logic
- Address Comparator
- Serial Clock Generator

Figure 229 • I²C Block Diagram



15.2.1.1 Input Glitch Filter

The I²C Fast mode (400 Kbit/s) specification states that glitches 50 ns or less should be filtered out of the incoming clock and data lines. The input glitch filter performs this function by filtering glitches on incoming clock and data signals. Glitches shorter than the glitch filter length are filtered out. The glitch filter length is defined in terms of APB interface clock cycles and configurable from 3 to 21 APB interface clock cycles. Input signals are synchronized with the internal APB interface clock (APB_0_CLK and APB_1_CLK).

Refer to the [Glitch Register](#), page 561 for more information on Glitch register bit definitions.

15.2.1.2 Arbitration and Synchronization Logic

In Master Mode, the arbitration logic monitors the data line. If any other device on the bus drives the data line Low, the I²C peripheral immediately changes from Master-Transmitter mode to Slave-Receiver mode. The synchronization logic synchronizes the serial clock generator block with the transmitted clock pulses coming from another master device.

The arbitration and synchronization logic implements the timeout requirements as per the SMBus specification version 2.0.

15.2.1.3 Address Comparator

When a master transmits a slave address on the bus, the address comparator checks the 7-bit slave address with its own slave address. If the transmitted slave address does not match, the address comparator compares the first received byte with the general call address (0x00). If the address matches, the Status register is updated. The general call address is used to address each device connected to the I²C-bus.

15.2.1.4 Serial Clock Generator

In Master mode, the serial clock generator generates the serial clock line (SCL). The clock generator is switched off when I²C is in Slave mode. Refer to the [I²C Clock Requirements](#), page 541 for details on the baud rate clock (BCLK).

15.2.2 Port List

The following table lists the various I²C signals. The letter X is used as a placeholder for 0 or 1 in register and signal descriptions.

Table 517 • I²C Interface Signals

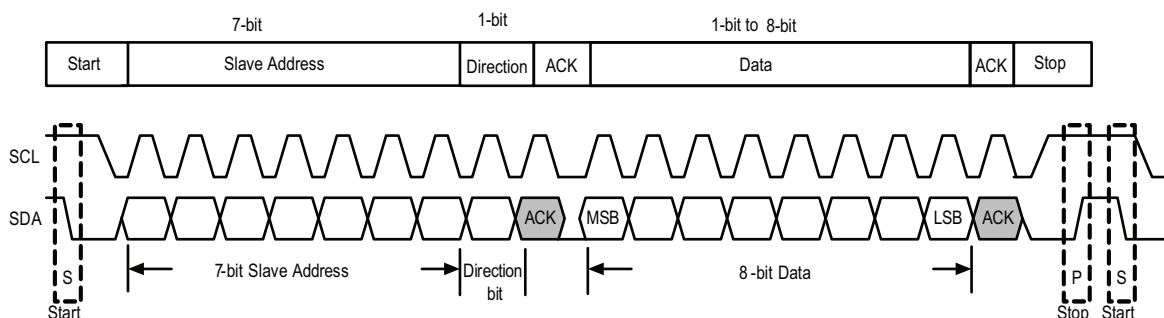
Name	Type	Polarity	Description
I2C_X_SCL	Input/Output	High	Serial clock
I2C_X_SDA	Input/Output	High	Serial data
I2C_X_SMBALERT_NI	Input	High	Input interrupt signal; used in Master mode to monitor whether the slave wants to force communication with the master.
I2C_X_SMBALERT_NO	Output	High	Output interrupt signal; used in Slave mode if the slave wants to force communication with the master.
I2C_X_SMBSUS_NI	Input	High	Input Suspend Mode signal; used in I ² C Slave mode.
I2C_X_SMBSUS_NO	Output	High	Output Suspend Mode signal; used in I ² C Master mode.

15.2.2.1 I²C Byte Transfer

A typical I²C 8-bit data transfer cycle is shown in the following figure. A start condition is signaled when the SDA line goes Low while the SCL line is High. After a start condition, the master transmits the 7-bit slave address followed by a direction bit, which is decoded and acknowledged (ACK) by the slave. Following the address phase, multiple bytes can be transferred with an ACK for each byte. The end of the transaction is signaled by a stop condition. The stop condition is signaled by the SDA line asserted High while the SCL line is High.

When the I²C peripheral is in a receiver (Master or Slave) mode, it may acknowledge or ignore the data sent by the transmitter. The I²C peripheral must send a no-acknowledge (NACK) bit during the acknowledge cycle on the bus to disable the data transfer by signaling the stop condition.

Figure 230 • 8-bit Data Transfer Cycle



15.2.3 Initialization

The I²C peripheral can be initialized in the user application software by configuring the I²C Control register and SOFT_RESET_CR system registry. The initialization sequence is as follows:

1. Release the I²C from reset by using SOFT_RESET_CR system registry ([Table 517](#), page 540).
2. Enable I²C by writing '1' to the ENS1 bit of [Control Register](#).
3. Configure the serial clock rate by using CR0, CR1, and CR2 bits of [Control Register](#).
4. Set the slave address to Slave0 Address Register ([Slave0 Address Register](#)).

15.2.3.1 I²C Reset

The I²C peripherals reset to zero on power-up and are held in reset until enabled. An option is provided under software control to reset the I²C peripherals by writing to bit 11 or bit 12 in the System register, SOFT_RESET_CR. The soft resets are encoded in the following table.

Table 518 • Soft Reset Bit Definitions For I²C Peripherals

Bit Number	Name	R/W	Reset Value	Description
12	I2C1_SOFTRESETR	R/W	1	Controls reset input to I2C_1 0: Release I2C_1 from reset 1: Keep I2C_1 in reset (reset value)
11	I2C0_SOFTRESET	R/W	1	Controls reset input to I2C_0 0: Release I2C_0 from reset 1: Keep I2C_0 in reset (reset value)

At power-up, the reset signals are asserted as 1. This keeps the I²C peripherals in a reset state. I²C peripheral becomes active when the bit is set to 0, as shown in [Table 518](#), page 541.

15.2.3.2 I²C Clock Requirements

The I2C_0 and I2C_1 peripherals are clocked by APB_0_CLK on APB bus 0 and APB_1_CLK on APB bus 1. These clocks are derived from the main MSS clock M3_CLK. Each APB clock can be programmed individually as M3_CLK divided by 1, 2, 4 or 8. Refer to the [UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide](#) for more information.

15.2.3.2.1 Baud Rate Clock

BCLK is a pulse-for-transmission speed control signal and is internally synchronized with the clock input. BCLK is used to set the serial clock frequency from a clock sourced within the FPGA fabric when the CR[2:0] bits in the Control register are set to 0b111(0x7). Otherwise, either APB_0_CLK or APB_1_CLK is used to determine the serial clock frequency. The actual non-stretched serial bus clock frequency can be calculated based on the settings in the CR[2:0] fields of the Control register and the frequencies of APB_0_CLK or APB_1_CLK and BCLK. Refer to the [Control Register](#), page 551 for more information on bit settings.

15.2.3.2.2 Clock Stretching

The I²C peripherals support the clock stretching feature as defined in Philips I²C v2.1 specifications. This addresses the condition where the I²C slave is unable to meet the clock speed provided by the I²C master and needs to slow down. Care should be taken so that the slowest I²C device does not dominate the bus performance. The I²C slave is allowed to hold down the clock if it needs to reduce the bus speed. The I²C master reads back the clock signal after releasing it to a High state and waits until the line goes High.

Clock stretching is a common practice. However, the total bandwidth of the shared bus might significantly decrease. Estimating the impact of clock stretching is required to share I²C bus among multiple devices.

15.2.4 Details of Operation

The I²C logic operates in the following modes:

- Master Mode
 - Master-Transmitter mode: The master transmits serial data on SDA and drives the SCL.
 - Master-Receiver mode: The master receives serial data on SDA and drives the SCL.
- Slave Mode
 - Slave-Receiver mode: Serial data and the serial clock are received through SDA and SCL.
 - Slave-Transmitter mode: Serial data is transmitted through SDA while the master drives SCL.

15.2.4.1 Master Mode

When the Cortex-M3 processor or any other bus master becomes the master, the I²C peripheral waits until the serial bus is free. When the serial bus is free, the I²C peripheral generates a start condition, sends the slave address, and transfers the direction bit. The I²C peripheral operates as a master transmitter or as a master receiver, depending on the transfer direction bit.

15.2.4.1.1 Transfer Example

1. The Cortex-M3 processor sets the ENS1 and STA bits of the Control register.
2. The I²C peripheral sends a START condition and then generates an interrupt request, (STATUS register = 0x08).
3. The Cortex-M3 processor writes to the data register (7-bit slave address and direction bit) and then clears the serial interrupt (SI) bit in the Control register.
4. The I²C peripheral sends the data register contents and then generates the interrupt request.
5. The Status register contains a value of 0x18 or 0x20, depending on the received ACK bit (Table 524, page 552).
6. The transfer is continued according to the STATUS Register – Master-Transmitter Mode.

15.2.4.2 Slave Mode

After setting the ENS1 bit in the [Control Register](#), the I²C peripheral is in Slave mode (which is not addressed by the master). I²C peripheral checks for its own slave address and the general call address. If one of these addresses is detected, the I²C peripheral is addressed by the master and then an interrupt is requested. The I²C peripheral can operate as a slave transmitter or a slave receiver.

15.2.4.2.1 Transfer Example

1. The Cortex-M3 processor sets the ENS1 and AA bits of the Control register.
2. The I²C peripheral receives its own address and the direction bit from master.
3. The I²C peripheral generates an interrupt request, Status register = 0x60 (Table 526, page 554).
4. The Cortex-M3 processor prepares to receive the data and then clears the SI bit in Control register.
5. The I²C peripheral receives the next data byte and generates the interrupt request. The Status register contains a value of 0x80 or 0x88, depending on the AA bit. Refer to Table 526, page 554.
6. The transfer is continued according to the STATUS Register – Slave-Receiver Mode.

15.2.4.3 SMBus and PMBus Overview

The SMBus is a two wire interface through which devices can communicate with each other. The SMBus interface can operate as a master or a slave. It is derived from the principles of operation of I²C. Refer to the [SMBus protocol v2.0 specification](#) for more information.

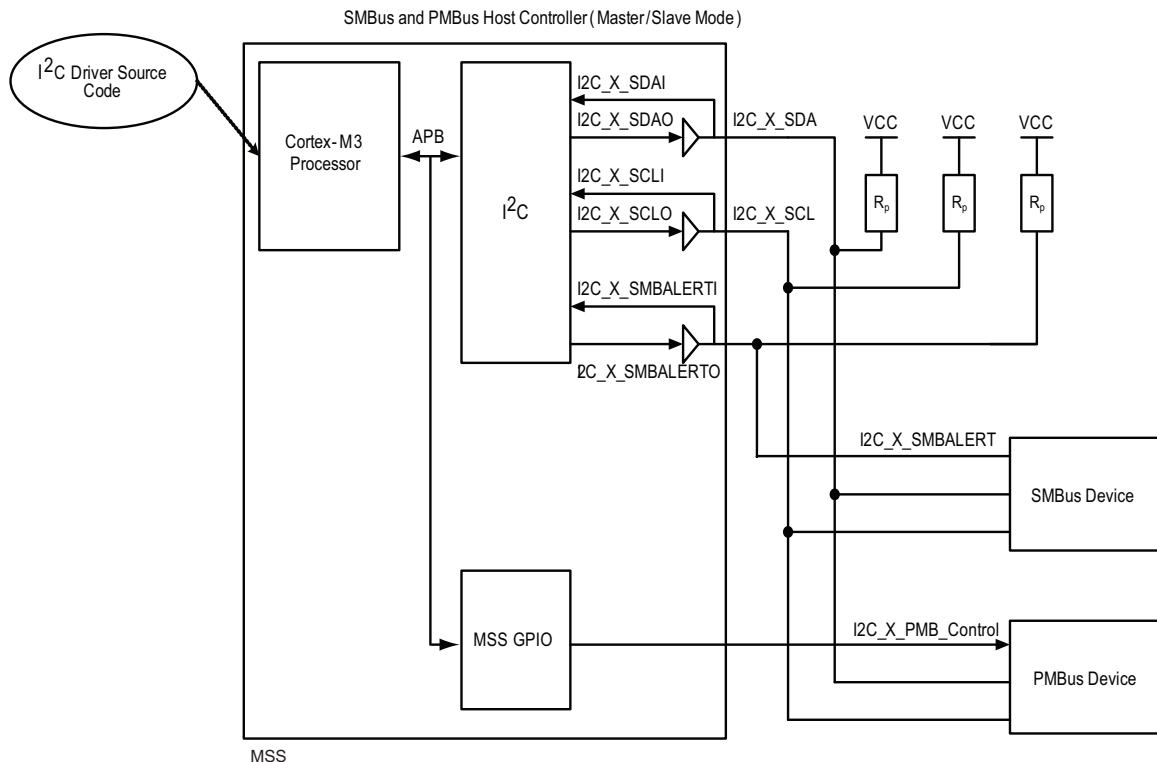
Power management bus (PMBus) is an open standard two wire communications protocol through which devices can communicate with each other. Refer to the [PMBus protocol v1.1 specification](#) for more information.

The following figure shows a PMBus or SMBus device interface example using the Cortex-M3 processor, I²C, MSS, and general purpose input/output (GPIO).

PMBus protocols are run through the serial bus, and the additional PMBus control signal is routed through the MSS GPIO. External SMBus devices may also be connected to the same bus, as shown in the figure.

For an SMBus application, it is advised to choose a PCLK so that the SCL transfers data at near the maximum frequency to ensure that other potential clock-stretching devices on the bus do not slow the clock frequency to below the minimum allowed SMBus clock frequency.

Figure 231 • PMBus and SMBus Devices Interface



15.2.4.4 I²C Interrupts

There are three interrupt signals for each I²C peripheral (I2C_0_INT, I2C_0_SMBALERT, and I2C_0_SMBSUS). These signals are generated by MSS I2C_0 and are mapped to INTISR 4, INTISR 5, and INTISR 6 in the Cortex-M3 processor NVIC controller. The I2C_1_INT, I2C_1_SMBALERT, and I2C_1_SMBSUS signals are generated by MSS I2C_1 and are mapped to INTISR 7, INTISR 8, and INTISR 9 in the Cortex-M3 processor NVIC controller. All interrupt enable bits within the NVIC, INTISR 4 through INTISR 9, correspond to bit locations 4 through 9.

Enable SMBus interrupts (I2C_X_SMBALERT and I2C_X_SMBSUS) in the I²C peripheral by setting the appropriate bits in the SMBUS register and clear the appropriate bit in the SMBus register in the interrupt service routine to prevent a reassertion of the interrupt.

The I2C_X_INT, I2C_X_SMBALERT, and I2C_X_SMBSUS I²C interrupt signals can be monitored by the FPGA logic through the fabric interface interrupt controller (FIIC). Refer to [Table 766](#), page 740 for further details.

15.3 How to Use I²C

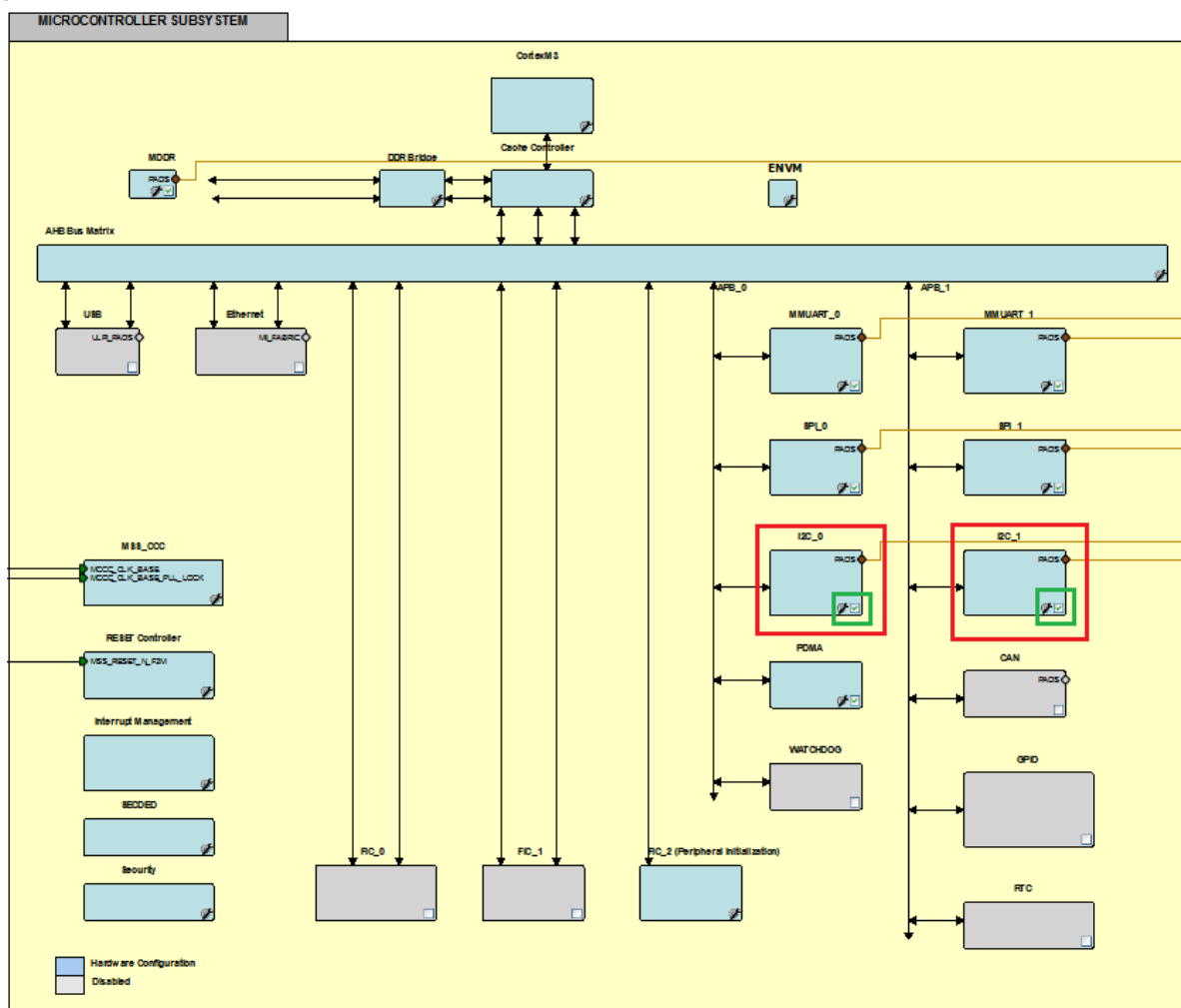
This section describes how to use I²C in an application.

15.3.1 Design Flow

The following steps are used to enable the I²C in the application by using Libero SoC.

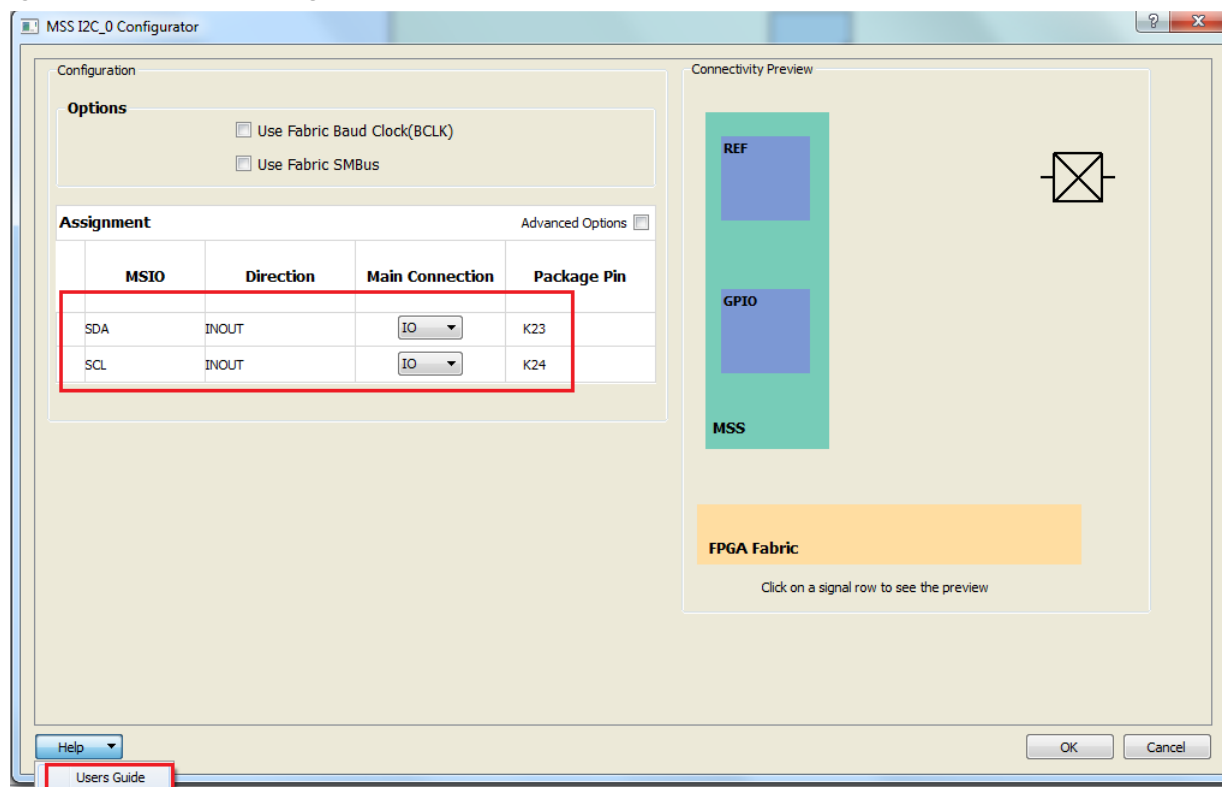
1. Enable I2C_0 and/or I2C_1 instance by using the MSS configurator in the application, as shown in the following figure.

Figure 232 • Enable I²C



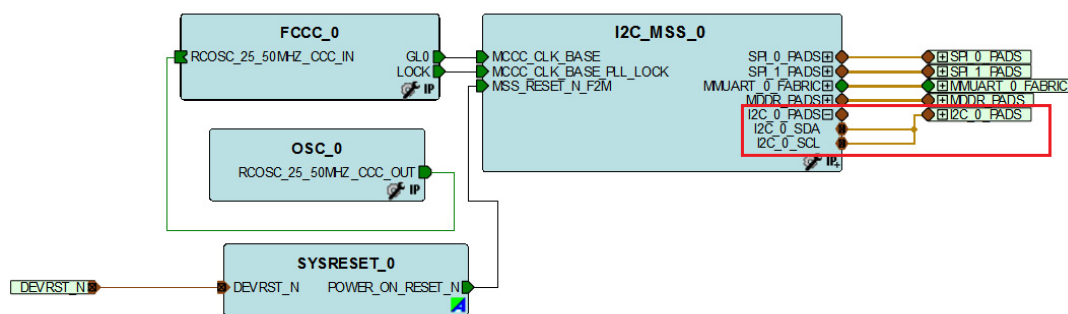
- Configure the ports of enabled I2C_0 instance to an IO by using MSS I2C_0 Configurator as shown in the following figure. Click the highlighted **Users Guide** button to find more information on I²C configuration details.

Figure 233 • MSS I2C Configurator

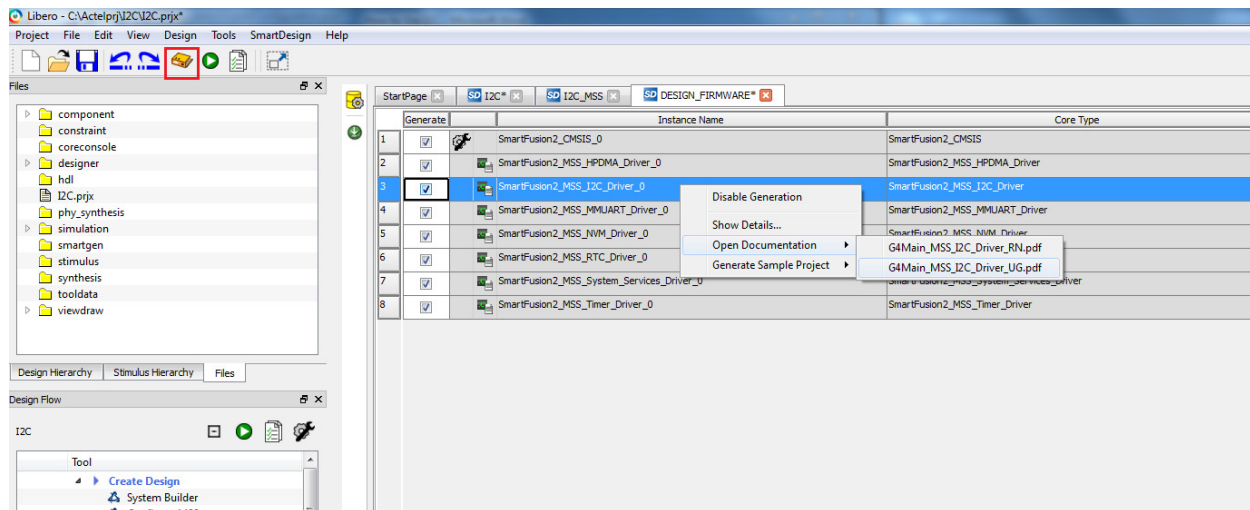


- The I2C_0 interface signals in the MSS component are shown in the following figure.

Figure 234 • I²C Interface Signals



4. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component**. For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace are included into the project. Click **Configure firmware**, as shown in the following figure to find the I²C driver information.

Figure 235 • I²C Driver User's Guide


5. Click **Generate Bitstream** under **Program Design** to complete *.fdb file generation.
6. Double-click **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_i2c firmware driver. The firmware driver, mss_i2c (mss_i2c.c and mss_i2c.h) which provides a set of functions for controlling the MSS I2Cs can also be downloaded from the Microsemi firmware catalog. The following figure lists the main APIs for I²C. For more information on the APIs, refer to the **SmartFusion2_MSS_I2C_Driver_UG**, as shown in the preceding figure.

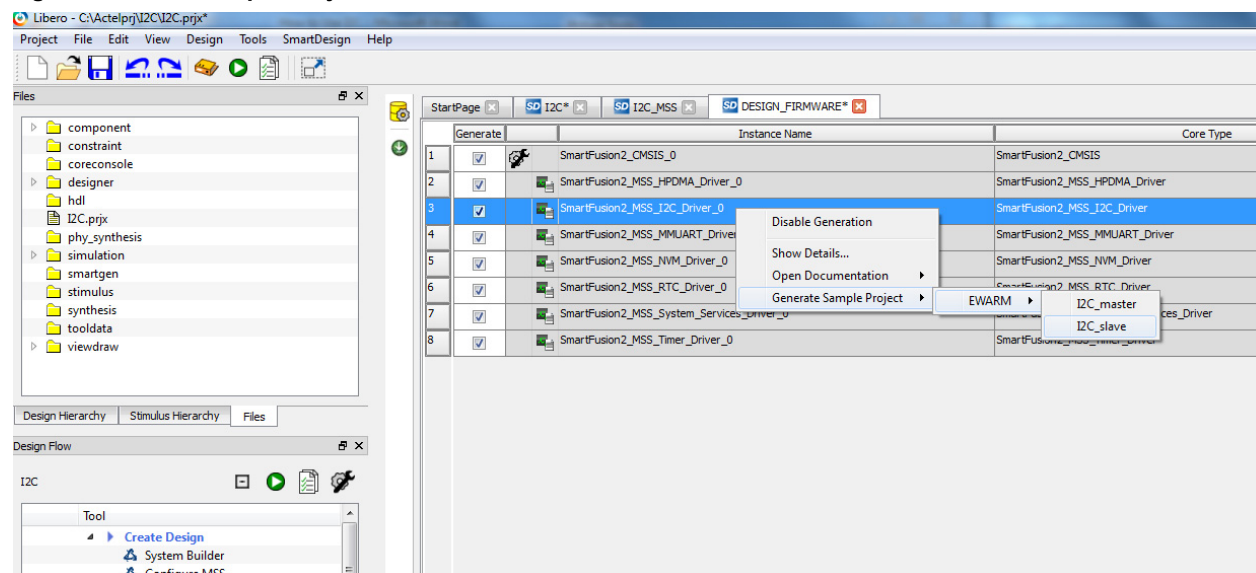
Table 519 • MSS I²C APIs

Category	API	Description and Usage
Initialization and configuration function	MSS_I2C_init()	Initializes and configures the I ² C with MSS I2C's configuration as parameters
I ² C master operation functions	MSS_I2C_write()	Initiates an I ² C master write transaction
	MSS_I2C_read()	Initiates an I ² C master read transaction
	MSS_I2C_write_read()	Initiates an I ² C write-read transaction
	MSS_I2C_wait_complete()	Waits for the current I ² C transaction to complete
	MSS_I2C_get_status()	Gets the current state of an MSS I2C instance
I ² C slave operation functions	MSS_I2C_set_slave_tx_buffer()	Specifies the memory buffer holding the data that will be sent to the I ² C master
	MSS_I2C_set_slave_rx_buffer()	Specifies the memory buffer that will be used by the MSS I2C slave instance to receive data
	MSS_I2C_set_slave_mem_offset_length()	Specifies the number of bytes expected as part of the write phase of a write-read transaction
	MSS_I2C_register_write_handler()	Registers the function that is called to process the data written to MSS I2C instance when it is the slave in an I ² C write transaction
	MSS_I2C_enable_slave()	Enables the MSS I2C slave
	MSS_I2C_disable_slave()	Disables the MSS I2C slave

Table 519 • MSS I²C APIs (continued)

Category	API	Description and Usage
SMBus control functions	MSS_I2C_smbus_init()	Initializes SMBus timeouts and status logics
	MSS_I2C_suspend_smbus_slave()	Forces slave devices into power-down or suspend mode
	MSS_I2C_set_smbus_alert()	Used to force master communication by an I ² C slave device
	MSS_I2C_enable_smbus_irq()	Enables interrupt related to SMBus which can be either SMBSUS or SMBALERT interrupt
	MSS_I2C_disable_smbus_irq()	Disables interrupt related to SMBus which can be either SMBSUS or SMBALERT interrupt

7. For more information on I²C usage, the sample projects are available and can be generated, as shown in the following figure.

Figure 236 • I²C Sample Project

Note: The MSS I2C does not support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for more information.

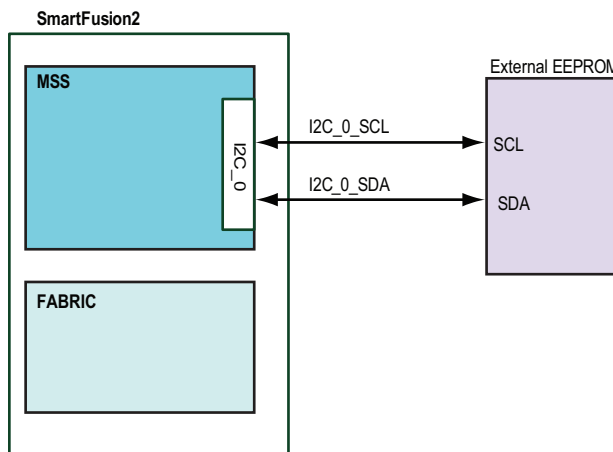
15.3.2 I²C Use Models

The following sections describe I²C use models.

15.3.2.1 Use Model 1: Interfacing External EEPROM

The following figure shows the interfacing of the external EEPROM to I2C_0 of the SmartFusion2 MSS.

Figure 237 • Interfacing External EEPROM to MSS I2C_0 - Block Diagram



Refer to the [Design Flow](#), page 544 to configure I2C_0 in the application. EEPROM is the target slave device in this example.

15.3.2.1.1 Software Design Flow

I²C Master Mode

The I²C instance I2C_0 can be initialized by using MSS_I2C_init API. Refer to the **SmartFusion2_MSS_I2C_Driver_UG** to use the I²C initialization API.

Write Operation

Write data to the target slave device using MSS_I2C_write API. This API ensures the presence of I2C_0 in Master mode for write transactions. The following parameters are required to use the I²C write API:

- Target slave device address (Ex.EEPROM)
- Data which is to be written to the target device
- Data size in bytes

Read Operation

Read data from the target slave device using MSS_I2C_read API. This API ensures the presence of I2C_0 in Master mode for read transactions. The following parameters are required to use the I²C read API:

- Target slave device address (Ex.EEPROM)
- Data buffer to collect the data from the target device
- Data size in bytes

For any I²C read or write transaction to complete, MSS_I2C_wait_complete API should be called. This API waits for the current I²C transaction to complete.

15.3.2.2 Use Model 2: Configuring I²C as a Slave

The I²C peripheral can be configured to Slave mode. In Slave mode, the I2C_0 responds to the commands received by the I²C master device.

Refer to the [Design Flow](#), page 544 to configure I2C_0 in the application.

15.3.2.2.1 Software Design Flow

I²C Slave Mode

The I²C instance I2C_0 can be initialized by using the MSS_I2C_init API. The slave device address should be specified while initializing.

Write Operation

Following are the steps to complete the write transaction:

- Set the slave receive buffer
The data receive buffer is used to store the data received when the I²C slave is the target of an I²C write transaction. Use the MSS_I2C_set_slave_rx_buffer API.
- Register the write handler
The handler function must be called on completion of the I²C write transaction. MSS_I2C_register_write_handler API can be used to register the write handler function.
- Enable the slave
The MSS_I2C_enable_slave API can be used to enable the slave.

Read Operation

Following are the steps to complete the read transaction:

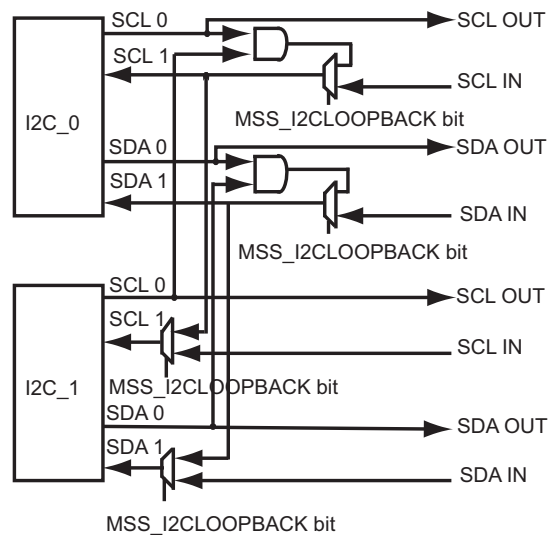
- Set the slave transmit buffer
The data buffer is transmitted when the I²C slave is the target of an I²C read transaction. Here, use the MSS_I2C_set_slave_tx_buffer API.
- Enable the slave
MSS_I2C_enable_slave API can be used to enable the slave.

15.3.2.3 Use Model 3: I²C Loopback Mode

I2C_0 and I2C_1 are internally connected in Loopback mode, as shown in the following figure. The MSS_I2CLOOPBACK bit of the LOOPBACK_CR System register is used to enable the Loopback mode.

Following are the steps to configure I2C_0 and I2C_1 in Loopback mode to verify the data transfer between I²C peripherals.

- Enable the I2C_0 and I2C_1 in the MSS configurator of the Libero SoC design project.
- Initialize I2C_0 and I2C_1 in the SoftConsole application.
- Set the MSS_I2CLOOPBACK bit of the LOOPBACK_CR System register in the SoftConsole application. For further details, refer to the [System Register Block](#), page 670.
- Use I²C read and write APIs from the MSS I²C driver to send and receive the data. While using APIs from the MSS I²C driver, ensure that either I2C_0 or I2C_1 is in Master mode.
- The data traffic from I2C_0 is looped back to I2C_1, and vice versa.

Figure 238 • I²C Loopback Block Diagram

15.4 I²C Register Map

The internal register address map and reset values of each APB accessible register for I²C peripherals are listed in the following table. The I2C_0 base address resides at 0x40002000 and extends to address 0x40002FFF in the Cortex-M3 processor memory map. The I2C_1 base address resides at 0x40012000 and extends to address 0x40012FFF in the Cortex-M3 processor memory map.

Table 520 • I²C Register Map

Register Name	Address Offset	R/W	Reset Value	Description
CTRL	0x00	R/W	0x00	Control register: Used to configure the I ² C peripheral.
STATUS	0x04	R	0xF8	Status register: Read-only value which indicates the current state of the I ² C peripheral.
DATA	0x08	R/W	0x00	Data register: Read/write data to/from the serial interface.
SLAVE0 ADR	0x0C	R/W	0x00	Slave0 address register: Contains the primary programmable address of the I ² C peripheral.
SMBUS	0x10	R/W	0b01X1X000	SMBus register: Configuration register for SMBus timeout reset condition and for the optional SMBus signals SMBALERT_N and SMSBUS_N.
FREQ	0x14	R/W	0x08	Frequency register: Necessary for configuring real-time timeout logic. Can be set to the PCLK frequency for 25 ms SMBus timeouts, or the timeout value maybe increased/decreased.
GLITCHREG	0x18	R/W	0x03	Glitch Reg length register: Used to adjust the input glitch filter length. If GLITCHREG_FIXED = 0, then the register can be set from 3 to 21.
SLAVE1 ADR	0x1C	R/W	0x00	Slave1 address register: Contains the secondary programmable address of the I ² C peripheral.

15.4.1 Control Register

The following table describes the Control register used for configuring the I²C peripherals.

Table 521 • Control Register (CTRL)

Bit Number	Name	R/W	Reset Value	Description
7	CR2	R/W	0	Clock rate bit 2; refer to bit 0.
6	ENS1	R/W	0	Enable bit. When ENS1 = 0, the SDA and SCL outputs are in a high impedance and SDA and SCL input signals are ignored. When ENS1 = 1, the I ² C is enabled.
5	STA	R/W	0	The Start flag. When STA = 1, the I ² C peripheral checks the status of the serial bus and generates a START condition, if the bus is free. STA bit is automatically cleared after START condition has been generated.
4	STO	R/W	0	The Stop flag. When STO = 1 and the I ² C is in Master mode, a STOP condition is transmitted to the serial bus. STO bit is automatically cleared after STOP condition has been generated.
3	SI	R/W	0	The SI flag. The SI flag is set by the I ² C whenever there is a serviceable change in the Status register. Once the register is updated, the SI bit must be cleared by software. The SI bit is directly readable through the APB INTERRUPT signal.
2	AA	R/W	0	The assert acknowledge flag. When AA = 1, an acknowledge is returned when: – The own slave address is received – The general call address is received when the GC bit in the address register is set – A data byte is received when the core is in the Master-receiver mode – A data byte is received when the core is in the Slave-receiver mode. When AA = 0, a not acknowledge is returned when: – A data byte is received while the core is in Master-receiver mode – A data byte is received when I ² C peripheral is in Slave-receiver mode
1	CR1	R/W	0	Serial clock rate bit 1; refer to bit 0
0	CR0	R/W	0	Serial clock rate bit 0. Clock rate is defined in Table 522 , page 551. BCLK is synchronized to PCLK and hence must be PCLKFREQ/2 or less.

Table 522 • Clock Rate (CR)

CR2	CR1	CR0	SCL Frequency
0	0	0	PCLK frequency/256
0	0	1	PCLK frequency/224
0	1	0	PCLK frequency/192
0	1	1	PCLK frequency/160
1	0	0	PCLK frequency/960
1	0	1	PCLK frequency/120
1	1	0	PCLK frequency/60
1	1	1	BCLK frequency/8

15.4.2 Status Register

The following table describes the Status register of the I²C peripherals.

Table 523 • Status Register (STATUS)

Bit Number	Name	R/W	Reset Value	Description
7:0	Status register	R	0XF8	The Status register is read-only. The status values depend on the mode of operation. They are listed in Table 524 , page 552 through Table 528 on page 558. Whenever there is a change of state, interrupt is requested. After updating any registers, the APB interface control must clear the interrupt by clearing the SI bit of the Control Register (CTRL) register.

15.4.2.1 Status Register: Master-Transmitter Mode

Table 524 • Status Register – Master-Transmitter Mode

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0x08	A START condition is transmitted.	Load SLA+W		0	0		SLA+W is transmitted; ACK is received
0x10	A repeated START condition is transmitted.	Load SLA+W		0	0		SLA+W is transmitted; ACK is received
		Load SLA+R		0	0		SLA+R is transmitted; Core is switched to MST/REC mode.
0x18	SLA+W is transmitted; ACK is received.	Load data byte	0	0	0		Data byte is transmitted; ACK is received.
		No action	1	0	0		Repeated START is transmitted
			0	1	0		STOP condition is transmitted; STO flag is reset.
			1	1	0		STOP condition followed by a START condition is transmitted; STO flag is reset.
0x20	SLA+W is transmitted; not ACK (NACK) is received.	Load data byte	0	0	0		Data byte is transmitted; ACK is received.
		No action	1	0	0		Repeated START is transmitted
			0	1	0		STOP condition is transmitted; STO flag is reset.
			1	1	0		STOP condition followed by a START condition is transmitted; STO flag is reset.
0x28	Data byte in Data Register is transmitted; ACK is received.	Load data byte	0	0	0		Data byte is transmitted; ACK bit is received.
		No action	1	0	0		Repeated START is transmitted.
			0	1	0		STOP condition is transmitted; STO flag is reset.
			1	1	0		STOP condition followed by a START condition is transmitted; STO flag is reset.

Table 524 • Status Register – Master-Transmitter Mode (continued)

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0x30	Data byte in Data Register is transmitted; not ACK (NACK) is received.	Data byte	0	0	0		Data byte is transmitted; ACK is received.
		No action	1	0	0		Repeated START is transmitted.
			0	1	0		STOP condition is transmitted; STO flag is reset.
			1	1	0		STOP condition followed by a START condition is transmitted; STO flag is reset.
0x38	Arbitration lost in SLA+R/W or data bytes.	No action	0	0	0		The bus is released; not-addressed Slave mode is entered.
			1	0	0		A START condition is transmitted when the bus gets free.
0xD0	SMBus master reset is activated.	No action					Wait 35 ms for interrupt to be set, clear interrupt and proceed to F8H state.

Notes:

- SLA = Slave address
- SLV = Slave
- REC = Receiver
- TRX = Transmitter
- SLA+W = Master sends slave address then writes data to slave
- SLA+R = Master sends slave address then reads data from slave

15.4.2.2 Status Register: Master-Receiver Mode**Table 525 • STATUS Register – Master-Receiver Mode**

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0x08	A START condition is transmitted.	Load SLA+R		0	0		SLA+R is transmitted; ACK is received
0x10	A repeated START condition is transmitted.	Load SLA+R		0	0		SLA+R is transmitted; ACK is received
		Load SLA+W		0	0		SLA+W is transmitted; I ² C is switched to MST/TRX mode.
0x38	Arbitration lost in not ACK (NACK) bit.	No action	0	0	0		The bus is released; I ² C enters the Slave mode.
			1	0	0		A start condition is transmitted when the bus gets free.
0x40	SLA+R has been transmitted; ACK is received.	No action	0	0	0	0	Data byte is received; not ACK (NACK) is returned.
			0	0	0	1	Data byte is received; ACK is returned
0x48	SLA+R is transmitted; not ACK (NACK) is received.	No action	1	0	0		Repeated START condition is transmitted
			0	1	0		STOP condition is transmitted; STO flag is reset.
			1	1	0		STOP condition followed by a START condition is transmitted; STO flag is reset.

Table 525 • STATUS Register – Master-Receiver Mode (continued)

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0x50	Data byte has been received; ACK is returned.	Read data byte	0	0	0	0	Data byte is received; not ACK(NACK) is returned.
		Read data byte	0	0	0	1	Data byte is received; ACK is returned
0x58	Data byte is received; not ACK (NACK) is returned.	Read data byte	1	0	0		Repeated START condition is transmitted
		Read data byte	0	1	0		STOP condition is transmitted; STO flag is reset.
		Read data byte	1	1	0		STOP condition followed by a START condition is transmitted; STO flag is reset.
0xD0	SMBus master reset is activated.	No action			0		Wait 35 ms for interrupt to set, clear interrupt and proceed to F8H state.

Notes:

- SLA = Slave address
- SLV = Slave
- REC = Receiver
- TRX = Transmitter
- SLA+W = Master sends slave address then writes data to slave
- SLA+R = Master sends slave address then reads data from slave

15.4.2.3 Status Register: Slave-Receiver Mode**Table 526 • STATUS Register – Slave-Receiver Mode**

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0x60	Own SLA+W is received; ACK is returned.	No action		0	0	0	Data byte is received and not ACK (NACK) is returned.
				0	0	1	Data byte is received and ACK is returned.
0x68	Arbitration lost in SLA+R/W as master; own SLA+W is received, ACK returned.	No action		0	0	0	Data byte is received and not ACK (NACK) is returned.
				0	0	1	Data byte is received and ACK is returned.
0x70	General call address (00H) is received; ACK is returned.	No action		0	0	0	Data byte is received and not ACK (NACK) is returned.
				0	0	1	Data byte is received and ACK is returned.
0x78	Arbitration lost in SLA+R/W as master; general call address is received, ACK returned.	No action		0	0	0	Data byte is received and not ACK (NACK) is returned.
				0	0	1	Data byte is received and ACK is returned.
0x80	Previously addressed with own SLV address; DATA is received; ACK returned.	Read data byte		0	0	0	Data byte is received and not ACK (NACK) is returned.
				0	0	1	Data byte is received and ACK is returned.

Table 526 • STATUS Register – Slave-Receiver Mode (continued)

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0x88	Previously addressed with own SLA; DATA byte is received; not ACK (NACK) returned.	Read data byte	0	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address.
			0	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized.
			1	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address; START condition is transmitted when the bus gets free.
			1	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized; START condition is transmitted when the bus gets free.
0x90	Previously addressed with general call address; DATA is received; ACK returned.	Read data byte		0	0	0	Data byte is received and not ACK (NACK) is returned.
				0	0	1	Data byte is received and ACK is returned.
0x98	Previously addressed with general call address; DATA is received; not ACK (NACK) returned.	Read data byte	0	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address.
			0	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized.
			1	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address; START condition is transmitted when the bus gets free.
			1	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized; START condition is transmitted when the bus gets free.

Table 526 • STATUS Register – Slave-Receiver Mode (continued)

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0xA0	A STOP condition or repeated START condition is received while addressed as SLV/REC or SLV/TRX.	No action	0	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address.
			0	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized.
			1	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address; START condition is transmitted when the bus gets free.
			1	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized; START condition is transmitted when the bus gets free.
0xD8	25 ms SCL low time is reached; device must be reset.	No action		X	0		Slave must proceed to reset state by clearing the interrupt within 10ms, according to SMBus specification v2.0.

Notes:

- SLA = Slave address
- SLV = Slave
- REC = Receiver
- TRX = Transmitter
- SLA+W = Master sends slave address then writes data to slave
- SLA+R = Master sends slave address then reads data from slave

15.4.2.4 Status Register – Slave-Transmitter Mode**Table 527 • STATUS Register – Slave-Transmitter Mode**

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0xA8	Own SLA+R is received; ACK is returned.	Load data byte		0	0	0	Last data byte is transmitted; ACK is received
				0	0	1	Data byte is transmitted; ACK is received
0xB0	Arbitration lost in SLA+R/W as master; own SLA+R is received; ACK is returned.	Load data byte		0	0	0	Last data byte is transmitted; ACK is received
				0	0	1	Data byte is transmitted; ACK is received
0xB8	Data byte is transmitted; ACK is received.	Load data byte		0	0	0	Last data byte is transmitted; ACK is received
				0	0	1	Data byte is transmitted; ACK is received

Table 527 • STATUS Register – Slave-Transmitter Mode (continued)

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0xC0	Data byte is transmitted; not ACK (NACK) is received.	No action	0	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address.
			0	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized.
			1	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address; START condition is transmitted when the bus gets free.
			1	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized; START condition is transmitted when the bus gets free.
0xC8	Last data byte is transmitted; ACK is received.	No action	0	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address.
			0	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized.
			1	0	0	0	Switched to not-addressed SLV mode; no recognition of own SLA or general call address; START condition is transmitted when the bus gets free.
			1	0	0	1	Switched to not-addressed SLV mode; own SLA or general call address is recognized; START condition is transmitted when the bus gets free.
0xD8	25 ms SCL low time is reached; device must be reset.	No action			0		Slave must proceed to reset state by clearing the interrupt within 10 ms, according to SMBus specification v2.0.

Notes:

- SLA = Slave address
- SLV = Slave
- REC = Receiver
- TRX = Transmitter
- SLA+W = Master sends slave address then writes data to slave
- SLA+R = Master sends slave address then reads data from slave

15.4.2.5 Status Register: Miscellaneous States

Table 528 • STATUS Register – Miscellaneous States

Status Code	Status	Data Register Action	Control Register Bits				Next Action Taken by Core
			STA	STO	SI	AA	
0x38	Arbitration lost	No action	0	0	0		Bus is released
			1	0	0		A Start condition is transmitted when the bus gets free.
0xF8	No relevant state information available; SI = 0	No Action	No Action				Idle
0x00	Bus error during MST or selected Slave modes	No action	0	1	0		Only the internal hardware is affected in the MST or addressed SLV modes. In all cases, the bus is released and the state switched in non-addressed Slave mode. Stop flag is reset.

15.4.3 Data Register

The Data register contains a byte of serial data to be transmitted or a byte that is received. The Cortex-M3 processor or any other fabric master can read from and write to this 8-bit, directly addressable register when it is not shifting a byte (after an interrupt is generated).

The bit descriptions are provided in the following table, in both data and addressing context. Data context is the 8-bit data format from MSB to LSB. Addressing context is based on a master sending an address call to a slave on the bus, along with a direction bit (master transmits data or receives data from a slave).

Table 529 • Data Register (DATA)

Bit Number	Name	R/W	Reset Value	Description
7	SD7	R/W	0	Data context: serial data bit 7(MSB) Addressing context: serial address bit 6(MSB)
6	SD6	R/W	0	Data context: serial data bit 6 Addressing context: serial address bit 5
5	SD5	R/W	0	Data context: serial data bit 5 Addressing context: serial address bit 4
4	SD4	R/W	0	Data context: serial data bit 4 Addressing context: serial address bit 3
3	SD3	R/W	0	Data context: serial data bit 3 Addressing context: serial address bit 2
2	SD2	R/W	0	Data context: serial data bit 2 Addressing context: serial address bit 1
1	SD1	R/W	0	Data context: serial data bit 1 Addressing context: serial address bit 0 (LSB)
0	SD0	R/W	0	Data context: serial data bit 0 (LSB) Addressing context: direction bit. 0 = Write; 1 = Read

15.4.4 Slave0 Address Register

The I²C has dual slave address (Slave0/Slave1) decoding capability. The Slave0 address register is a read/write directly accessible register. The details of this register are provided in the following table.

Table 530 • Slave0 Address Register (Slave0 ADR)

Bit Number	Name	R/W	Reset Value	Description
7	ADR6	R/W	0	Own Slave0 address bit 6
6	ADR5	R/W	0	Own Slave0 address bit 5
5	ADR4	R/W	0	Own Slave0 address bit 4
4	ADR3	R/W	0	Own Slave0 address bit 3
3	ADR2	R/W	0	Own Slave0 address bit 2
2	ADR1	R/W	0	Own Slave0 address bit 1
1	ADR0	R/W	0	Own Slave0 address bit 0
0	GC	R/W	0	General call (GC) address acknowledge. If the GC bit is set, the general call address is recognized; otherwise it is ignored.

15.4.5 SMBus Register

The I²C SMBus is an optional bus for serial data transfer between the MSS and the FPGA fabric for Suspend mode. Suspend mode is a Low power mode where most devices are stalled or powered down. The SMBus register contains specific SMBus related functionality and signals. The details of this register are provided in the following table.

Table 531 • SMBus Register (SMBUS)

Bit Number	Name	R/W	Reset Value	Description
7	SMBus reset	R/W	0	Writing one to this bit forces the clock line Low until 35 ms is exceeded, thus resetting the entire bus as per the SMBus specification v2.0. Usage: When the I ² C is used as a host controller (master), reset the bus by holding the clock line Low 35 ms. Slaves must react to this event and reset themselves.
6	SMBSUS_NO control	R/W	0b1	SMBSUS_NO control. SMBUS_NO is a Suspend mode signal from MSS to fabric. It is used in Master/Host mode to force other devices into Power down/Suspend mode. It is active low signal. SMBSUS_NO and SMBSUS_NI are separate signals (not wired-AND). If the I ² C is part of a host-controller, SMBSUS_NO can be used as an output; if I ² C is a slave to a host-controller that is implemented SMBSUS_N, then only SMBSUS_NI's status is relevant.

Table 531 • SMBus Register (SMBUS) (continued)

Bit Number	Name	R/W	Reset Value	Description
5	SMBSUS_NI status	R	0bX	Status of SMBSUS_NI signal. SMBUS_NI is a Suspend mode signal from fabric to MSS. It is used if the core is slave/device. Upon resuming, the SMBSUS_NI returns High. The system then returns all devices to their operational state. SMBSUS_NO and SMBSUS_NI are separate signals (not wired-AND). If the I ² C is part of a host-controller, SMBSUS_NO can be used as an output; if I ² C is a slave to a host-controller that is implemented SMBSUS_N, then only SMBSUS_NI's status is relevant.
4	SMBALERT_NO control	R/W	0b1	SMBALERT_NO control; SMBALERT_NO is a wired-and-interrupt signal from the MSS to fabric. It is used in Slave/Device mode if the core wants to force communication with a host.
3	SMBALERT_NI status	R	0bX	Status of SMBALERT_NI signal. SMBALERT_NI is a wired-and-interrupt signal from the MSS to fabric. It is used in Master/Host mode, if the slave/devices want to force communication with a host.
2	SMBus enable	R/W	0	0: SMBus timeouts and status logic disabled (standard I ² C bus operation) 1: SMBus timeouts and status logic enabled
1	SMBSUS interrupt enable	R/W	0	0: SMBSUS interrupt signal (SMBS) disabled 1: SMBSUS interrupt signal (SMBS) enabled
0	SMBALERT interrupt enable	R/W	0	0: SMBALERT interrupt signal (SMBA) disabled 1: SMBALERT interrupt signal (SMBA) enabled

15.4.6 Frequency Register

The Frequency register is required to calculate the real-time timeout logic. The following table describes the Frequency register.

Table 532 • Frequency Register (FREQ)

Bit Number	Name	R/W	Reset Value	Description
7:0	Frequency	R/W	0x08	PCLKx frequency in MHz from 1 to 255. If the PCLKx frequency is used, and SMBus is enabled, the SMBus timeouts are configured per the SMBus specification. If another timeout value is desired, scale the frequency value as per the following formula: Timeout scale = Fscale/Factual If the actual PCLKx frequency is 100 MHz, and a scale down is desired that results in a 3 ms timeout rather than 25 ms timeout, then: Fscale = 3/25 x Factual = 0.12 x 100 = 12 MHz Writing 12 into the Frequency register has the effect of reducing maximum timeout count value and reducing the real-time timeout from 25 ms to 3 ms.

15.4.7 Glitch Register

The Glitch register (GLITCHREG) gives the size of the glitch (in terms of APB interface clock cycles) to filter the glitches on data and clock lines.

Table 533 • Glitch Register (GLITCHREG)

Bit Number	Name	R/W	Reset Value	Description
7:0	GlitchReg_Num	R/W	0x03	This read/write register is used to adjust the input glitch filter length. Depending on the application, the glitch filter is used to suppress spikes between 3 and 21 APB interface clock cycles. Number or length of shift register filter is set to value from 3 to 21.

15.4.8 Slave1 Address Register

The I²C has dual slave address (Slave0/Slave1) decoding capability. The Slave0 address register is a read/write directly accessible register. The details of this register are provided in the following table.

Table 534 • Slave1 Address Register (SLAVE1 ADR)

Bit Number	Name	R/W	Reset Value	Description
7	ADR6	R/W	0	Own Slave1 address bit 6
6	ADR5	R/W	0	Own Slave1 address bit 5
5	ADR4	R/W	0	Own Slave1 address bit 4
4	ADR3	R/W	0	Own Slave1 address bit 3
3	ADR2	R/W	0	Own Slave1 address bit 2
2	ADR1	R/W	0	Own Slave1 address bit 1
1	ADR0	R/W	0	Own Slave1 address bit 0
0	ENADR	R/W	0	1: Enable the Slave1 address comparisons 0: Disable Slave1 address comparisons

16 MSS GPIO

The microcontroller subsystem (MSS) general purpose input/output (GPIO) block is an advanced peripheral bus (APB) slave that provides access to 32 GPIOs. As shown in the following figure, MSS masters and fabric masters can access the MSS GPIO block through the advanced high-performance bus (AHB) matrix.

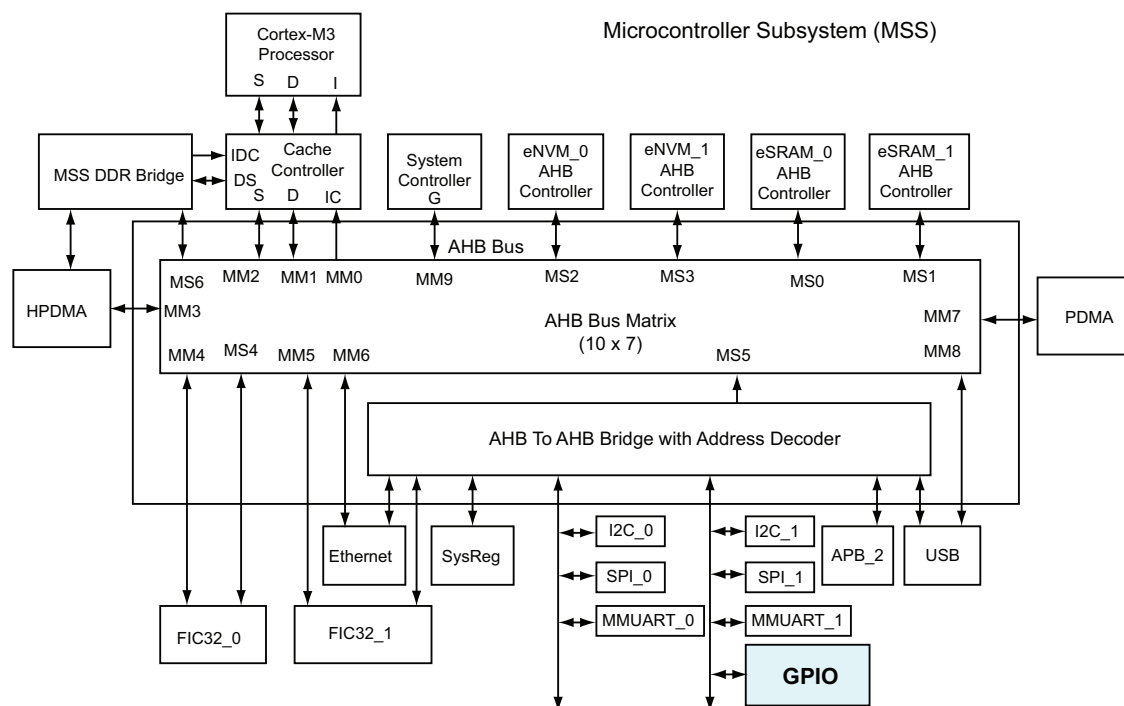
16.1 Features

Following are the features of the MSS GPIO block:

- 32 individually configurable GPIOs
- Each GPIO is dynamically programmable as an input, output, or bi-directional I/O.
- Each GPIO can be configured as an interrupt source to the ARM® Cortex® -M3 processor in Input mode
- The reset state of the GPIOs is configurable
- The GPIOs can be selectively reset by either the hard reset (power-on reset, user reset from the fabric) or the soft reset from the SYSREG block

The MSS GPIO block features mentioned above can be configured using Libero SoC software.

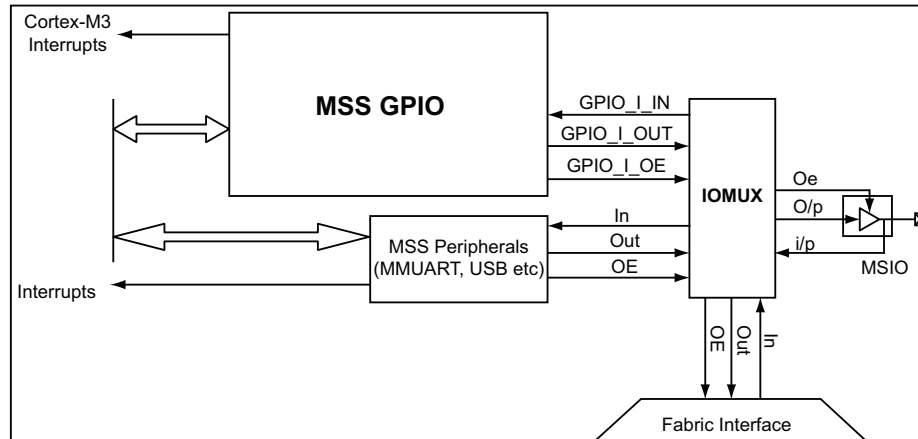
Figure 239 • GPIO Connected on APB Slave in MSS



16.2 MSS GPIO Functional Description

The following figure shows the internal architecture of the MSS GPIO block. GPIOs and MSS peripherals, such as MMUART, SPI, and I2C, can be routed to MSIO pads or to the field programmable gate array (FPGA) fabric through I/O multiplexers (MUXes), as shown in the figure.

Figure 240 • GPIO, IOMUX, and MSIO



The MSS GPIO block contains the following:

- 32-bit input register (GPIO_IN), which holds the input values
- 32-bit output register (GPIO_OUT), which holds the output values
- 32-bit interrupt register (GPIO_IRQ), which holds the interrupt state
- 32 configuration registers (GPIO_X_CFG), one register for each GPIO

When a GPIO is configured in Input mode, the GPIO input is passed through a 2 flip-flop synchronizer and latched into the GPIO_IN register. The GPIO_IN register value is read through the APB bus and is accessible to the Cortex-M3 processor or fabric master. The input to the GPIO can be from the fabric or MSIO pad.

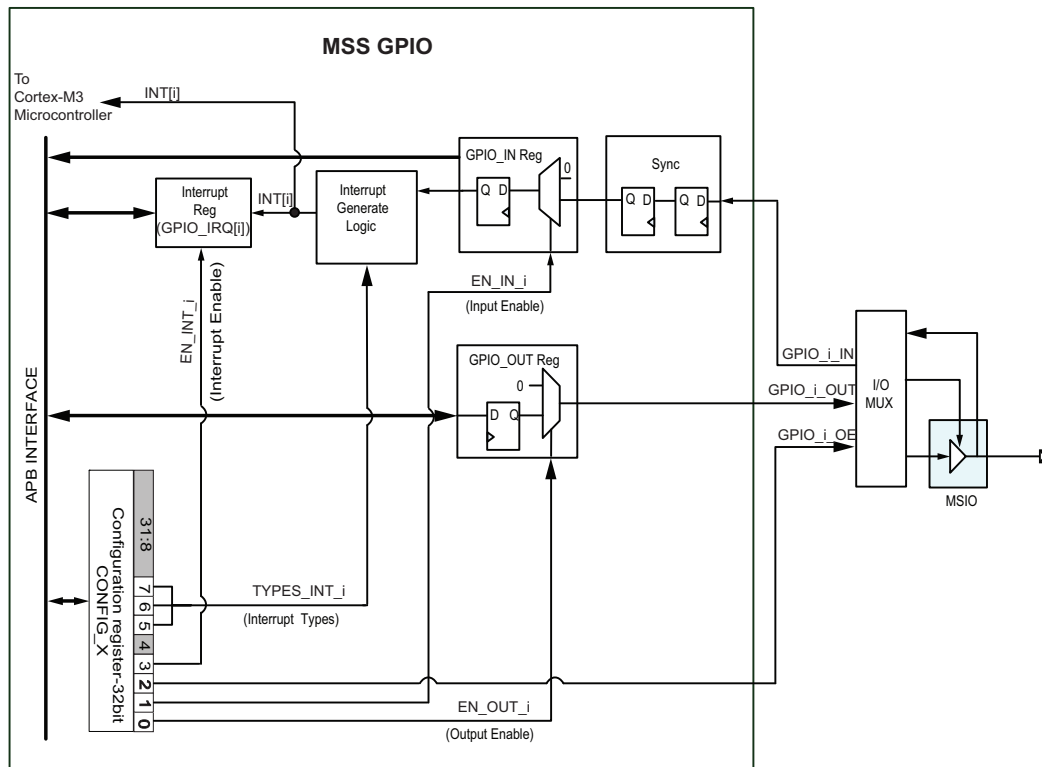
The GPIO_IN register output can also be used as an interrupt to the Cortex-M3 processor. This can be configured as an edge triggered (on rising edge, falling edge, or both edges) or as a level sensitive (active low or active high) interrupt. The interrupt is latched in the GPIO_IRQ register and is accessible through the APB bus, as shown in [Table 536](#), page 566.

In Edge-sensitive mode, GPIO_IRQ register is cleared either by disabling the interrupt or writing a logic 1 through the APB interface. If an edge and GPIO_IRQ clearing through the APB occurs simultaneously, the edge has higher priority.

[Table 536](#), page 566 maps the GPIO input to the nested vectored interrupt controller (NVIC) interrupt number in the Cortex-M3 processor. The interrupt input to the GPIO can be from the fabric or MSIO pad.

When the GPIO is configured in an Output mode, the output value can be configured using the APB bus and is accessible to the Cortex-M3 processor or fabric master. The GPIO output can be available to the MSIO pad, fabric, or the FPGA and MSIO pads.

For configuring GPIO in Input, Output, and Bi-directional modes, refer to the bit definitions in [Table 535](#), page 565.

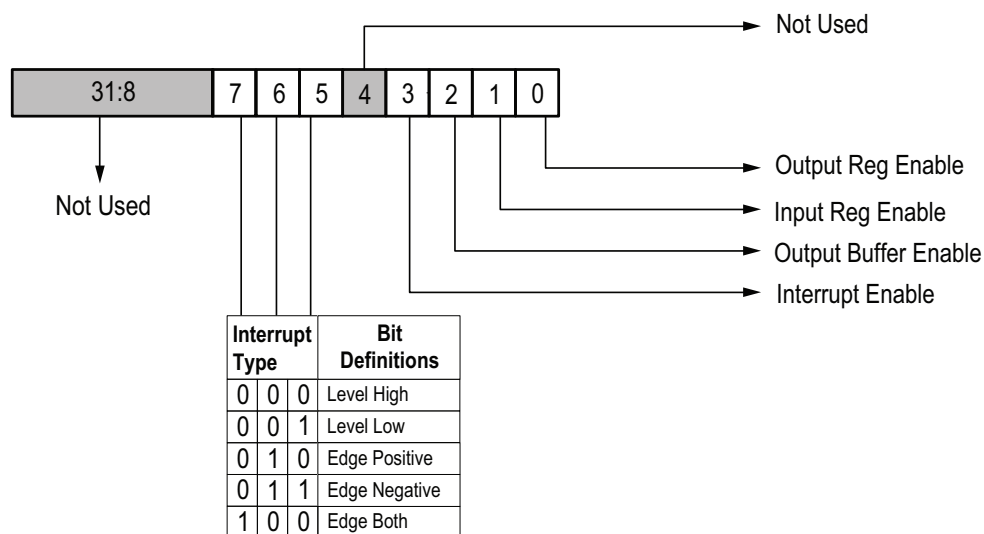
Figure 241 • MSS GPIO Block Diagram


16.2.1 MSS GPIO Configuration Registers (GPIO_X_CFG)

In the MSS GPIO block, each GPIO has a 32-bit configuration register. The configuration register allows selection of the GPIO in Input, Output, or Bi-directional mode. A GPIO can also be used as an interrupt when it is configured in Bi-directional mode. The offset address of MSS GPIO block configuration registers is given in [Table 538](#), page 574. The following figure shows the detailed configuration register bit definitions.

Figure 242 • GPIO Configuration Register (GPIO_X_CFG)

CONFIG_X (X = 0 to 31)



The following table provides bit definitions for the GPIO configuration registers.

Table 535 • GPIO_X_CFG

Bit Number	Name	Type	Reset Value	Description
GPIO_X_CFG[31:8]	Reserved	R/W	0h000000	Reserved
GPIO_X_CFG[7:5]	TYPES_INT_I	R/W	0b000	Input interrupt type configuration Refer Figure 242 , page 565 for Bit definitions.
GPIO_X_CFG[4]	Reserved	R/W	0b0	Reserved
GPIO_X_CFG[3]	EN_INT_i	R/W	0b0	0: Interrupt disabled 1: Interrupt enabled
GPIO_X_CFG[2]	GPIO_i_OE	R/W	0b0	0: Disable output buffer 1: Enable output buffer
GPIO_X_CFG[1]	EN_IN_i	R/W	0b0	0: Input register disabled 1: Input register enabled
GPIO_X_CFG[0]	EN_OUT_i	R/W	0b0	0: Output register disabled 1: Output register enabled

Table 536 • MSS GPIO Interrupts

Cortex-M3 Processor Interrupt	Signal	Source	Description
INTISR[50]	GPIO_INT[0]	GPIO_0	Interrupt from GPIO_0
INTISR[51]	GPIO_INT[1]	GPIO_1	Interrupt from GPIO_1
INTISR[52]	GPIO_INT[2]	GPIO_2	Interrupt from GPIO_2
INTISR[53]	GPIO_INT[3]	GPIO_3	Interrupt from GPIO_3
INTISR[54]	GPIO_INT[4]	GPIO_4	Interrupt from GPIO_4
INTISR[55]	GPIO_INT[5]	GPIO_5	Interrupt from GPIO_5
INTISR[56]	GPIO_INT[6]	GPIO_6	Interrupt from GPIO_6
INTISR[57]	GPIO_INT[7]	GPIO_7	Interrupt from GPIO_7
INTISR[58]	GPIO_INT[8]	GPIO_8	Interrupt from GPIO_8
INTISR[59]	GPIO_INT[9]	GPIO_9	Interrupt from GPIO_9
INTISR[60]	GPIO_INT[10]	GPIO_10	Interrupt from GPIO_10
INTISR[61]	GPIO_INT[11]	GPIO_11	Interrupt from GPIO_11
INTISR[62]	GPIO_INT[12]	GPIO_12	Interrupt from GPIO_12
INTISR[63]	GPIO_INT[13]	GPIO_13	Interrupt from GPIO_13
INTISR[64]	GPIO_INT[14]	GPIO_14	Interrupt from GPIO_14
INTISR[65]	GPIO_INT[15]	GPIO_15	Interrupt from GPIO_15
INTISR[66]	GPIO_INT[16]	GPIO_16	Interrupt from GPIO_16
INTISR[67]	GPIO_INT[17]	GPIO_17	Interrupt from GPIO_17
INTISR[68]	GPIO_INT[18]	GPIO_18	Interrupt from GPIO_18
INTISR[69]	GPIO_INT[19]	GPIO_19	Interrupt from GPIO_19
INTISR[70]	GPIO_INT[20]	GPIO_20	Interrupt from GPIO_20
INTISR[71]	GPIO_INT[21]	GPIO_21	Interrupt from GPIO_21
INTISR[72]	GPIO_INT[22]	GPIO_22	Interrupt from GPIO_22
INTISR[73]	GPIO_INT[23]	GPIO_23	Interrupt from GPIO_23
INTISR[74]	GPIO_INT[24]	GPIO_24	Interrupt from GPIO_24
INTISR[75]	GPIO_INT[25]	GPIO_25	Interrupt from GPIO_25
INTISR[76]	GPIO_INT[26]	GPIO_26	Interrupt from GPIO_26
INTISR[77]	GPIO_INT[27]	GPIO_27	Interrupt from GPIO_27
INTISR[78]	GPIO_INT[28]	GPIO_28	Interrupt from GPIO_28
INTISR[79]	GPIO_INT[29]	GPIO_29	Interrupt from GPIO_29
INTISR[80]	GPIO_INT[30]	GPIO_30	Interrupt from GPIO_30
INTISR[81]	GPIO_INT[31]	GPIO_31	Interrupt from GPIO_31

16.2.2 MSS GPIO Reset Functionality

The MSS GPIO outputs can be reset to a predefined state (logic 1 or logic 0) either by the soft resets coming from the SYSREG block or the power-on reset or the user reset signal (MSS_GPIO_RESET_N) from the FPGA fabric. The selection of the reset can be done through the MSS Configurator in Libero SoC or by writing to reset select registers of the SYSREG block. Refer to [Table 544](#), page 578 for reset select signals.

Following are the two reset resources of the MSS GPIO block:

- **Hard reset:**
 - The MSS_GPIO_RESET_N is a reset signal generated from the FPGA fabric
 - Power-on reset is a device reset signal
- **Soft reset from the SYSREG block:** There are two soft reset signals generated from the SYSREG block: one for the output registers and another for the input and interrupt registers.
 - The MSS_GPIO_SOFTRESET is a soft reset signal from the SYSREG block for the MSS GPIO block input registers (GPIO_IN) and interrupt registers (GPIO_IRQ)
 - There are four byte-wise soft reset signals from the SYSREG block to reset all the 32 GPIO output registers. Each soft reset signal resets the GPIO output byte (8-GPIO_OUT registers).
[Table 540](#), page 576 shows the soft reset signals from the SYSREG block

Each GPIO output byte reset is enabled by the control signals from the SYSREG block. The reset enable feature for each GPIO output byte is used to hold the GPIO_OUT register values from not getting affected by the reset source.

The GPIO output byte reset configuration in Libero SoC is explained in the [Initializing the MSS GPIO](#), page 570.

Note: If the byte reset enable signal (MSS_GPIO_X_Y_DEF) is enabled, the GPIO output byte (GPIO_OUT[X:Y]) can be reset by the soft reset signal or hard reset signal, depending upon the Reset select signal (MSS_GPIO_X_Y_SYSRESET_SEL).

16.3 MSS GPIO Usage

16.3.1 Configuring MSS GPIO Using Libero SoC

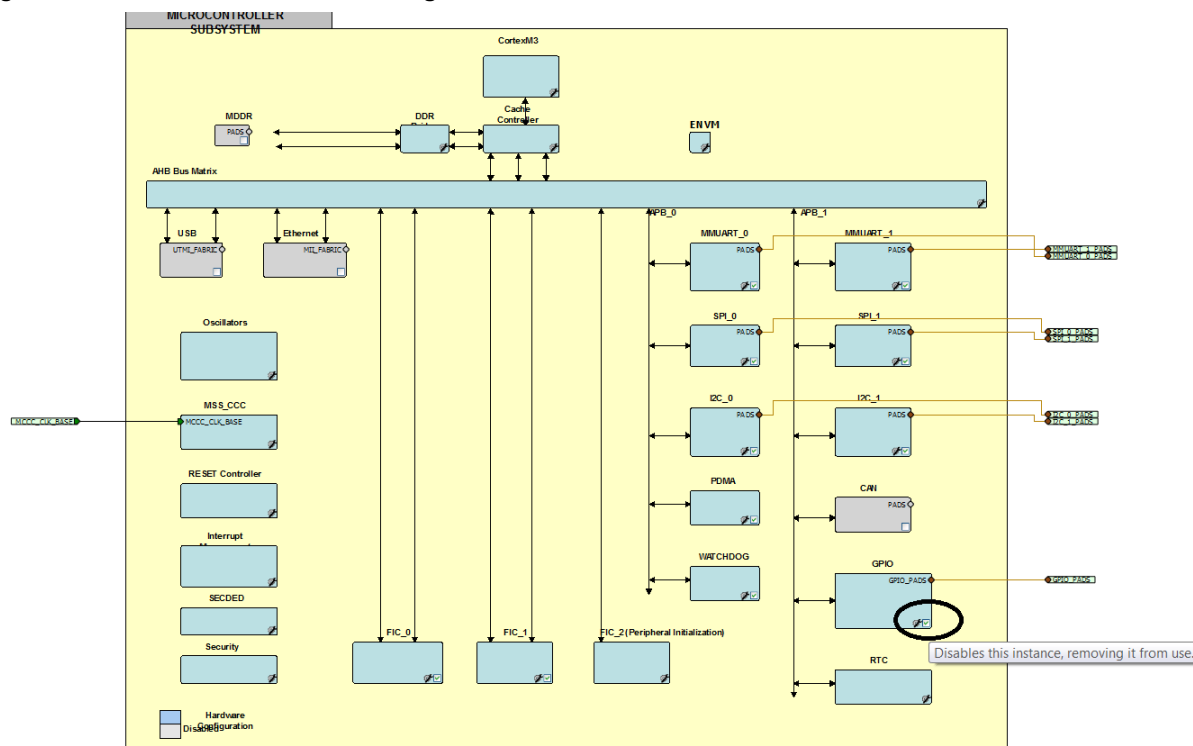
This sub-section describes MSS GPIO configuration in Libero SoC and shows different options available for configuring GPIO. The MSS GPIO is disabled by default in MSS configurator when the Libero SoC project is created.

Following pages describe the steps to be used to configure in Libero SoC.

Step 1

Enable the MSS GPIO in the Libero SoC project using MSS configurator, as shown in the following figure.

Figure 243 • Enable GPIO in MSS Configurator



Step 2

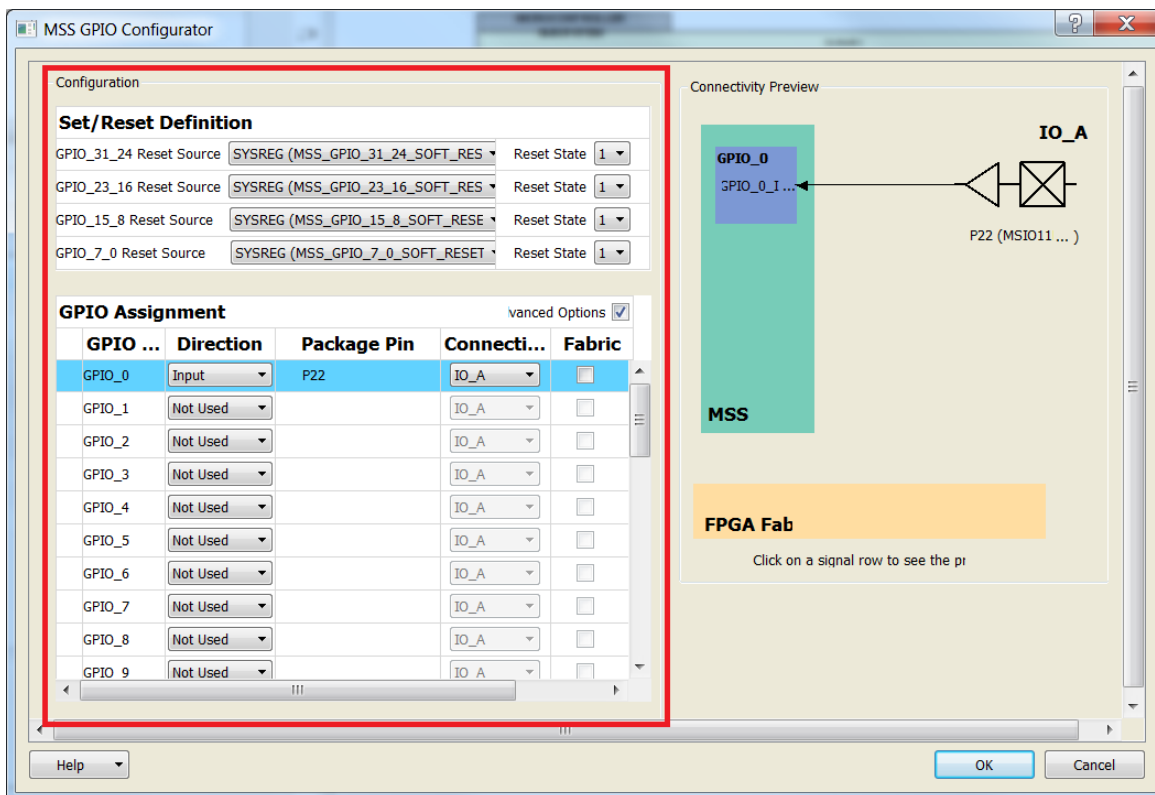
Double-click **GPIO** or right-click, **GPIO** and select the configure option to program the GPIOs.

16.3.1.1 GPIO Configurator Options

The following fields are shown in the GPIO configurator (see the following figure):

- Set/Reset definition: Used to initialize the GPIOs
- GPIO assignment: 3 programming fields associated with each GPIO
- Direction: Selects Input, Output, Bidirectional, or Tristate mode
- Package pin: Shows the MSIO pin associated with the GPIO
- Connectivity: Selects IO_A/ IO_B MSIO or FABRIC_A/ FABRIC_B
- Advanced options – When enabled, shows the possibility of connecting the GPIO input or GPIO output to the fabric.

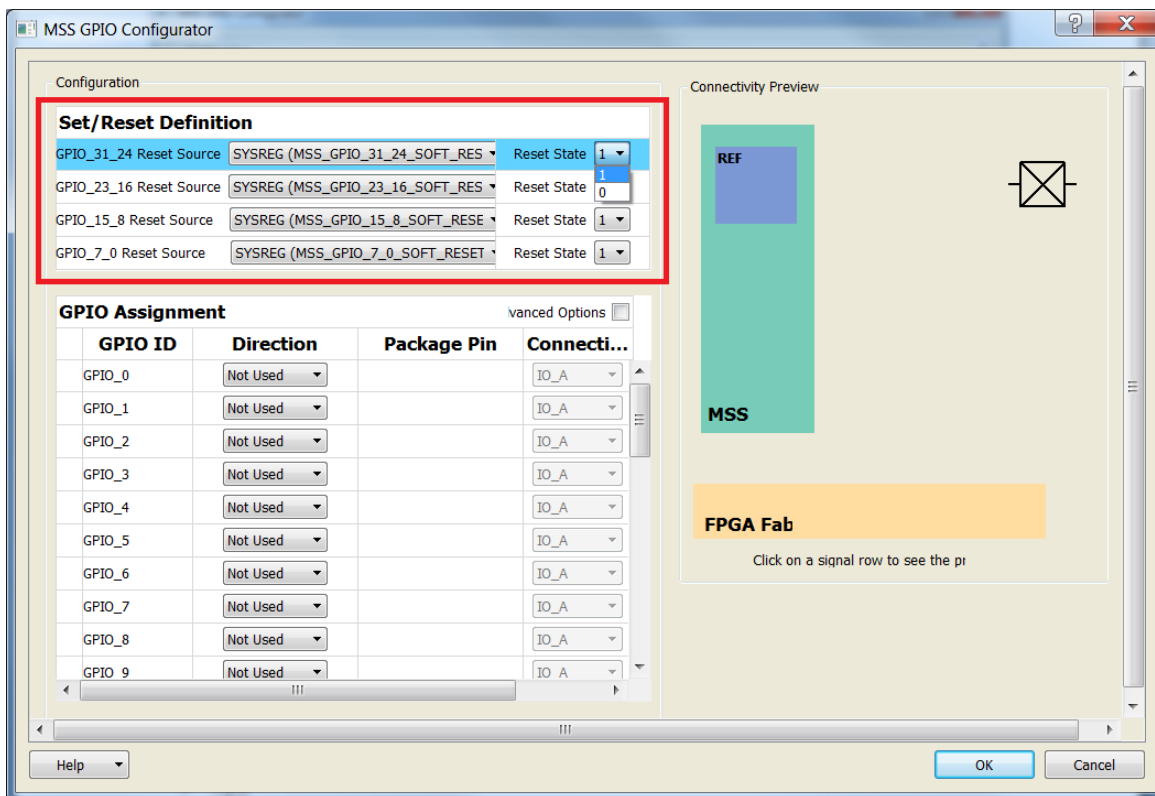
Figure 244 • MSS GPIO Configurator



16.3.1.2 Initializing the MSS GPIO

The highlighted section in the following figure shows the byte-wise initialization of GPIOs in Libero SoC.

Figure 245 • Configuring GPIO Byte-Wise Reset



16.3.1.3 Configuring MSS GPIOs as Input, Output, Tristate, and Bi-directional

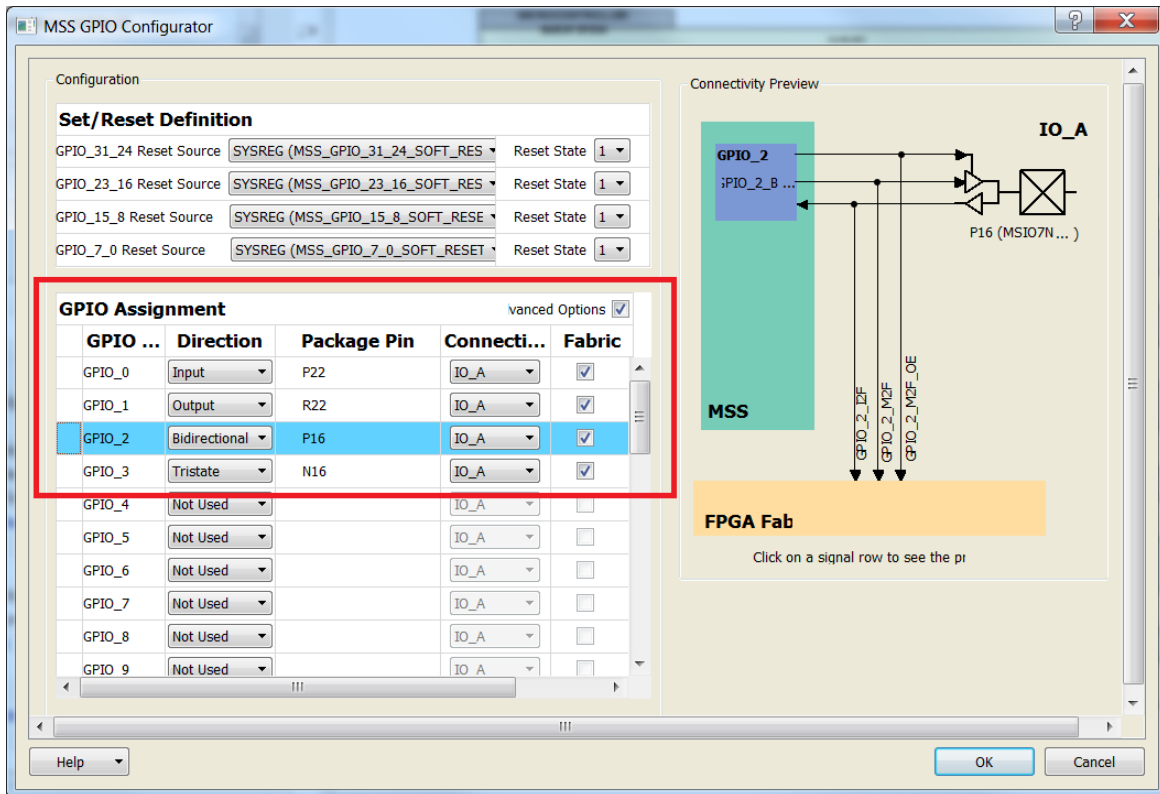
There are four modes of MSS GPIO, which can be configured using Libero SoC:

- Input
- Output
- Bi-directional
- Tristate

To configure a GPIO as an interrupt, configure GPIO as an input in Libero SoC. Enable the interrupt in SoftConsole using MSS GPIO application program interfaces (APIs). Refer to the [Use Model 1: Configuring GPIOs to Act as Interrupt to Cortex-M3 Processor](#), page 573.

To configure a GPIO to simultaneously connect with fabric, select the **Advanced Options** and **Fabric** check boxes in Libero SoC, as shown in the following figure.

Figure 246 • Configuring GPIOs as Input, Output, Tristate, or Bi-directional



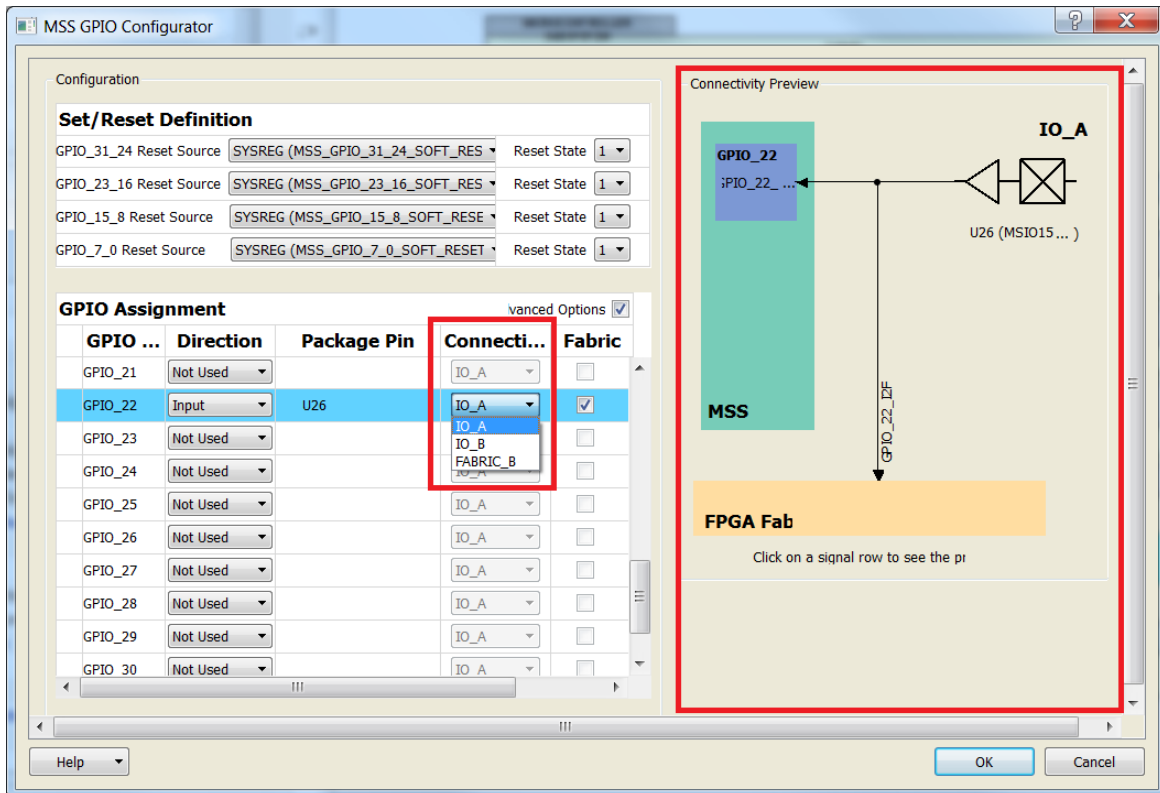
Connectivity Preview

GPIOs have access to IO_A or IO_B MSIOs. If an MSIO is used by any shared peripheral, GPIOs cannot access those particular MSIOs. GPIOs can be connected directly to the fabric by selecting FABRIC_A or FABRIC_B.

GPIOs connecting to MSIOs can also be shared with fabric using the **Advanced Options** of the MSS GPIO configurator.

The following figure shows a graphical view of the current connections for the highlighted GPIO signal.

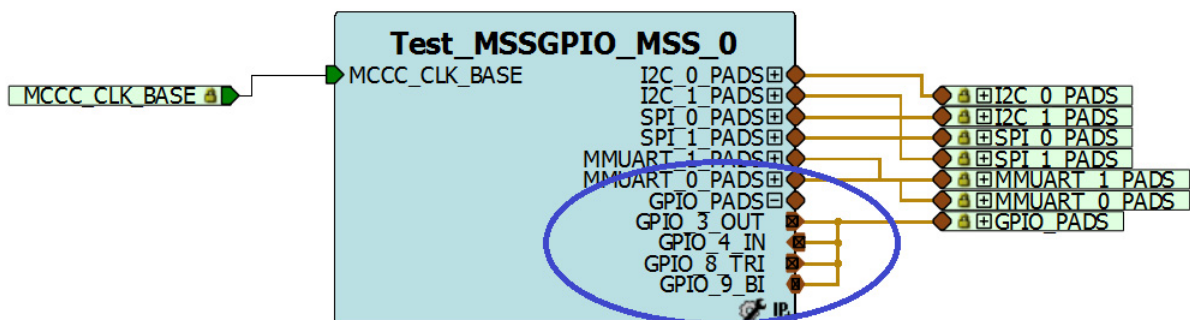
Figure 247 • Connectivity Preview



Step 3

The configured GPIO signals are shown in the following figure.

Figure 248 • GPIO Signals



Step 4

To generate a component, click the **Generate Component** shortcut in MSS configurator or select **SmartDesign > Generate Component**. The firmware driver folder and SoftConsole workspace will be included in the design project.

Step 5

Click **Generate Bitstream** under **Program Design** to complete the *.fdb file generation.

Step 6

Double-click **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_gpio firmware driver. The firmware driver mss_gpio (mss_gpio.h and mss_gpio.c), which provides a set of functions for controlling the MSS GPIOs, can be downloaded from the Microsemi Firmware Catalog.

Table 537 • MSS GPIO APIs

Category	API	Description and Usage
Initialization	MSS_GPIO_Init	Used for initializing the GPIO byte-wide reset state
Configuration	MSS_GPIO_config	Used to configure GPIO as input, output, tristate, or bidirectional
Reading and setting	MSS_GPIO_get_inputoutputs	Sets the GPIO input state
	MSS_GPIO_get_outputs	Get the GPIO input state
	MSS_GPIO_set_outputs	Sets the GPIO output state
	MSS_GPIO_set_inputs	Get the GPIO output state
Interrupt control	MSS_GPIO_disable_irq	Disable the GPIO interrupt feature
	MSS_GPIO_enable_irq	Enable the GPIO interrupt feature
	MSS_GPIO_clear_irq	Clear interrupt on GPIO

Note: The MSS GPIO supports full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for information.

16.3.2 MSS GPIO Use Models

This section explains the use models and gives directions for using MSS GPIOs in an application.

16.3.2.1 Use Model 1: Configuring GPIOs to Act as Interrupt to Cortex-M3 Processor

1. Enable GPIOs using MSS configurator in Libero SoC.
2. Initialize GPIOs to a predefined state using the MSS_GPIO_init API in SoftConsole or Libero SoC.
3. Configure GPIOs as inputs by using the MSS_GPIO_config API in SoftConsole or Libero SoC.
4. Use the interrupt API MSS_GPIO_enable_irq to dynamically configure the input from the MSIO to interrupt the Cortex-M3 processor. Use API MSS_GPIO_disable_irq to disable the interrupt to the Cortex-M3 processor.
5. Each GPIO input can also be read using the MSS_GPIO_get_inputs API.

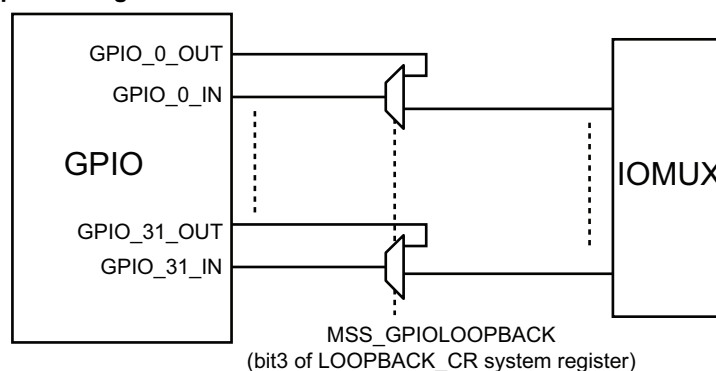
16.3.2.2 Use Model 2: GPIO Loopback Mode

GPIO Loopback mode (shown in the following figure) is looping back all the 32 GPIOs by controlling the MSS_LOOPBACK bit of LOOPBACK_CR System register.

1. Enable GPIOs using MSS configurator in Libero SoC.
 2. Initialize all 32 GPIOs to a predefined state using the MSS_GPIO_init API in SoftConsole or Libero SoC.
 3. Set the MSS_GPIOLOOPBACK1 bit of the LOOPBACK_CR System register in the SoftConsole application.
 4. Configure GPIOs as output and input by using the MSS_GPIO_config API.
 5. Use the MSS_GPIO_set_output API to set the value of the GPIO_OUT register.
 6. Use the interrupt API MSS_GPIO_enable_irq to configure all 32 GPIO outputs as an interrupts to the Cortex-M3 processor.
- (or)

Use the API MSS_GPIO_get_inputs to read the looped back values and use them the application.

Figure 249 • GPIO Loopback Diagram



16.4 GPIO Register Map

The base address of the MSS GPIO block is 0x40013000. The address offset of each MSS GPIO register is provided in the following table. The table also includes configuration, input, and output registers.

Table 538 • MSS GPIO Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
GPIO_0_CFG	0x00	R/W	0x0	Configuration register for GPIO 0
GPIO_1_CFG	0x04	R/W	0x0	Configuration register for GPIO 1
GPIO_2_CFG	0x08	R/W	0x0	Configuration register for GPIO 2
GPIO_3_CFG	0x0C	R/W	0x0	Configuration register for GPIO 3
GPIO_4_CFG	0x10	R/W	0x0	Configuration register for GPIO 4
GPIO_5_CFG	0x14	R/W	0x0	Configuration register for GPIO 5
GPIO_6_CFG	0x18	R/W	0x0	Configuration register for GPIO 6
GPIO_7_CFG	0x1C	R/W	0x0	Configuration register for GPIO 7
GPIO_8_CFG	0x20	R/W	0x0	Configuration register for GPIO 8
GPIO_9_CFG	0x24	R/W	0x0	Configuration register for GPIO 9
GPIO_10_CFG	0x28	R/W	0x0	Configuration register for GPIO 10
GPIO_11_CFG	0x2C	R/W	0x0	Configuration register for GPIO 11
GPIO_12_CFG	0x30	R/W	0x0	Configuration register for GPIO 12

Table 538 • MSS GPIO Register Map (continued)

Register Name	Address Offset	Register Type	Reset Value	Description
GPIO_13_CFG	0x34	R/W	0x0	Configuration register for GPIO 13
GPIO_14_CFG	0x38	R/W	0x0	Configuration register for GPIO 14
GPIO_15_CFG	0x3C	R/W	0x0	Configuration register for GPIO 15
GPIO_16_CFG	0x40	R/W	0x0	Configuration register for GPIO 16
GPIO_17_CFG	0x44	R/W	0x0	Configuration register for GPIO 17
GPIO_18_CFG	0x48	R/W	0x0	Configuration register for GPIO 18
GPIO_19_CFG	0x4C	R/W	0x0	Configuration register for GPIO 19
GPIO_20_CFG	0x50	R/W	0x0	Configuration register for GPIO 20
GPIO_21_CFG	0x54	R/W	0x0	Configuration register for GPIO 21
GPIO_22_CFG	0x58	R/W	0x0	Configuration register for GPIO 22
GPIO_23_CFG	0x5C	R/W	0x0	Configuration register for GPIO 23
GPIO_24_CFG	0x60	R/W	0x0	Configuration register for GPIO 24
GPIO_25_CFG	0x64	R/W	0x0	Configuration register for GPIO 25
GPIO_26_CFG	0x68	R/W	0x0	Configuration register for GPIO 26
GPIO_27_CFG	0x6C	R/W	0x0	Configuration register for GPIO 27
GPIO_28_CFG	0x70	R/W	0x0	Configuration register for GPIO 28
GPIO_29_CFG	0x74	R/W	0x0	Configuration register for GPIO 29
GPIO_30_CFG	0x78	R/W	0x0	Configuration register for GPIO 30
GPIO_31_CFG	0x7c	R/W	0x0	Configuration register for GPIO 31
GPIO_IRQ	0x80	R/W	0x0	Read Modify Write per interrupt bit.
GPIO_IN	0x84	R	0x0	GPIO input register – Read-only for input configured Ports.
GPIO_OUT	0x88	R/W	0x0	GPIO output register – Writeable/Readable for output configured ports. No action required for input configured ports.

16.4.1 SYSREG Block Registers

16.4.1.1 Register Map

The following table lists all the GPIO registers in the SYSREG block. The SYSREG block is located at address 0x40038000 in the Cortex-M3 processor address space.

Table 539 • GPIO SYREG Registers

Register Name	Address Offset	Register Type	Flash Write Protect	Reset Source	Description
MSS_GPIO_DEF	0x18C	RO-P		SYSRESET_N	MSS GPIO definition register
SOFT_RESET_CR	0x48	RW-P	Bit	SYSRESET_N	Generates software control interrupts to the MSS peripherals
LOOPBACK_CR	0x54	RW-P	Register	SYSRESET_N	Loopback control for MSS peripherals
GPIO_SYSRESET_SEL_CR	0x58	RW-P	Register	PORESET_N	Configures GPIO system reset.
GPIN_SRC_SEL_CR	0x5C	RW-P	Register	PORESET_N	Used to generate a GPIO input signal

16.4.2 Software Reset Control Register

Table 540 • SOFT_RESET_CR

Bit Number	Name	Reset Value	Description
24	MSS_GPOUT_31_24_SOFTRESET	0x1	0: Releases GPIO_OUT[31:24] from reset 1: Keeps GPIO_OUT[31:24] in reset
23	MSS_GPOUT_23_16_SOFTRESET	0x1	0: Releases GPIO_OUT[23:16] from reset 1: Keeps GPIO_OUT[23:16] in reset
22	MSS_GPOUT_15_8_SOFTRESET	0x1	0: Releases GPIO_OUT[15:8] from reset 1: Keeps GPIO_OUT[15:8] in reset
21	MSS_GPOUT_7_0_SOFTRESET	0x1	0: Releases GPIO_OUT[7:0] from reset 1: Keeps GPIO_OUT[7:0] in reset
20	MSS_GPIO_SOFTRESET	0x1	0: Releases the GPIO from reset, if not being held in reset by some other means 1: Keeps the GPIO held in reset Asserting this soft reset bit will hold the APB register, GPIO input, and interrupt generation logic in reset. GPIO OUT logic is not affected by this reset.

16.4.3 MSS GPIO Definitions

Table 541 • MSS_GPIO_DEF

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	Reserved
3	MSS_GPIO_31_24_DEF	0x1	Used to initialize GPIO bank [31:24] to 0 or 1 after reset.
2	MSS_GPIO_23_16_DEF	0x1	Used to initialize GPIO bank [23:16] to 0 or 1 after reset.
1	MSS_GPIO_15_8_DEF	0x1	Used to initialize GPIO bank [15:8] to 0 or 1 after reset.
0	MSS_GPIO_7_0_DEF	0x1	Used to initialize GPIO bank [7:0] to 0 or 1 after reset.

16.4.4 Loopback Control Register

Table 542 • LOOPBACK_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	Reserved
3	MSS_GPIOLOOPBACK	0	Controls whether internal loopback on the MSS GPIO is enabled. Allowed values: 0: No internal loopback for MSS GPIO 1: MSS GPIO outputs are looped back to MSS GPIO inputs

16.4.5 GPIO Input Source Select Control Register

Table 543 • GPIN_SRC_SEL_CR

Bit Number	Name	Reset Value	Description
[31:0]	MSS_GPINSOURCE	0	This 32-bit signal is used as select signal to generate a GPIO input signal by selecting two output signals from different IOMUXCELL or signals from I/O pads.

Note: The Libero automatically takes care of GPIO source input selection between current IOMUXCELL and alternate IOMUXCELL.

16.4.6 GPIO System Reset Control Register

Table 544 • GPIO_SYSRESET_SEL_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	Reserved
3	MSS_GPIO_31_24_SYSRESET_SEL	0	0: The GPIO[31:24] is reset by either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric. 1: The GPIO[31:24] is reset by the soft reset signal MSS_GPIO_31_24_SOFT_RESET.
2	MSS_GPIO_23_16_SYSRESET_SEL	0	0: The GPIO[23:16] is reset by either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric. 1: The GPIO[23:16] is reset by the soft reset signal MSS_GPIO_23_16_SOFT_RESET.
1	MSS_GPIO_15_8_SYSRESET_SEL	0	0: The GPIO[15:8] is reset by either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric. 1: The GPIO[15:8] is reset by the soft reset signal MSS_GPIO_15_8_SOFT_RESET.
0	MSS_GPIO_7_0_SYSRESET_SEL	0	0: The GPIO[7:0] is reset by either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric. 1: The GPIO[7:0] is reset by the soft reset signal MSS_GPIO_7_0_SOFT_RESET.

16.4.7 I/O MUX Associated With GPIOs

The following table shows the associated IOMUXes for GPIOs.

Table 545 • Associated IOMUXes for GPIOs

Peripheral	Associated IOMUXes
GPIOA[0] to GPIOA[10]	11 to 21
GPIOA[11] to GPIOA[18]	26 to 33
GPIOA[19] to GPIOA[22]	22 to 25
GPIOA[23] to GPIOA[24]	34 to 35
GPIOB[11] to GPIOB[16]	36 to 41
GPIOB[17] to GPIOB[22]	45 to 50
GPIOB[23]	56
GPIOB[24] to GPIOB[26]	42 to 44
GPIOB[27] to GPIOB[31]	51 to 55

The following tables ([Table 546](#), page 579 through [Table 591](#) on page 591) describe the IOMUXes in which GPIOs are shared with the peripheral and the fabric.

16.4.7.1 IOMUX CELL 11

I/O pad name: I2C1_SDA_USBA_DATA3_MGPIO0A

Table 546 • IOMUX CELL 11

I/O Pad Ports	IOMUXCELL Interface				
	I2C_1	USB Controller	GPIO[0]	FPGA Fabric	
IN	SDA_1_IN	USBA_DATAI[3]	GPIOA_IN[0]	IN_A	F2H_GPIN[11]
OUT	GND	DATAO[3]	GPIO_OUT[0]	IN_B	F2H_SCP[11]
OE	~SDA_1_OUT	~NDATAOE	GPIO_OE[0]	OUT_A	H2F_A[11]
				OUT_B	H2F_B[11]

16.4.7.2 IOMUX CELL 12

I/O pad name: I2C1_SCL_USBA_DATA4_MGPIO1A

Table 547 • IOMUX CELL 12

I/O Pad Ports	IOMUXCELL Interface				
	I2C_1	USB Controller	GPIO[1]	FPGA Fabric	
IN	SCL_1_IN	USBA_DATAI[4]	GPIOA_IN[1]	IN_A	F2H_GPIN[12]
OUT	GND	DATAO[4]	GPIO_OUT[1]	IN_B	F2H_SCP[12]
OE	~SCL_1_OUT	~NDATAOE	GPIO_OE[1]	OUT_A	H2F_A[12]
				OUT_B	H2F_B[11]

16.4.7.3 IOMUX CELL 13

I/O pad name: CAN_TXBUS_USBA_DATA0_MGPIO2A

Table 548 • IOMUX CELL 13

I/O Pad Ports	IOMUXCELL Interface				
	CAN	USB Controller	GPIO[2]	FPGA Fabric	
IN	NC	USBA_DATAI[0]	GPIOA_IN[2]	IN_A	F2H_GPIN[13]
OUT	CAN_TX_BUS	DATAO[0]	GPIO_OUT[2]	IN_B	F2H_SCP[13]
OE	VDD	~NDATAOE	GPIO_OE[2]	OUT_A	H2F_A[13]
				OUT_B	H2F_B[13]

16.4.7.4 IOMUX CELL 14

I/O pad name: CAN_RXBUS_USBA_DATA1_MGPIO3A

Table 549 • IOMUX CELL 14

I/O Pad Ports	IOMUXCELL Interface				
	CAN	USB Controller	GPIO[3]	FPGA Fabric	
IN	CAN_RX_BUS	USBA_DATAI[1]	GPIOA_IN[3]	IN_A	F2H_GPIN[14]
OUT	GND	DATAO[1]	GPIO_OUT[3]	IN_B	F2H_SCP[14]
OE	GND	~NDATAOE	GPIO_OE[3]	OUT_A	H2F_A[14]
				OUT_B	H2F_B[14]

16.4.7.5 IOMUX CELL 15

I/O pad name: CAN_TX_EBL_USBA_DATA2_MGPIO4A

Table 550 • IOMUX CELL 15

I/O Pad Ports	IOMUXCELL Interface				
	CAN	USB Controller	GPIO[4]	FPGA Fabric	
IN	NC	USBA_DATAI[2]	GPIOA_IN[4]	IN_A	F2H_GPIN[15]
OUT	CAN_TB_EBL_n	DATAO[2]	GPIO_OUT[4]	IN_B	F2H_SCP[15]
OE	VDD	~NDATAOE	GPIO_OE[4]	OUT_A	H2F_A[15]
				OUT_B	H2F_B[15]

16.4.7.6 IOMUX CELL 16

I/O pad name: SPI0_SDI_USBA_DIR_MGPIO5A

Table 551 • IOMUX CELL 16

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[5]	FPGA Fabric	
IN	SPISDI_0_IN	USBA_DIR	GPIOA_IN[5]	IN_A	F2H_GPIN[16]
OUT	GND	GND	GPIO_OUT[5]	IN_B	F2H_SCP[16]
OE	GND	GND	GPIO_OE[5]	OUT_A	H2F_A[16]
				OUT_B	H2F_B[16]

16.4.7.7 IOMUX CELL 17

I/O pad name: SPI0_SDO_USBA_STP_MGPIO6A

Table 552 • IOMUX CELL 17

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[6]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[6]	IN_A	F2H_GPIN[17]
OUT	SPISDO_0_OUT	STP	GPIO_OUT[6]	IN_B	F2H_SCP[17]
OE	SPIOEN_0	VDD	GPIO_OE[6]	OUT_A	H2F_A[17]
				OUT_B	H2F_B[17]

16.4.7.8 IOMUX CELL 18

I/O pad name: SPI0_SS0_USBA_NXT_MGPIO7A

Table 553 • IOMUX CELL 18

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[7]	FPGA Fabric	
IN	SPISSI_0_IN	USBA_NXT	GPIOA_IN[7]	IN_A	F2H_GPIN[18]
OUT	SPISS_0_OUT[0]	GND	GPIO_OUT[7]	IN_B	F2H_SCP[18]
OE	SPI_0_MASTER	GND	GPIO_OE[7]	OUT_A	H2F_A[18]
				OUT_B	H2F_B[18]

16.4.7.9 IOMUX CELL 19

I/O pad name: SPI0_SS1_USBA_DATA5_MGPIO8A

Table 554 • IOMUX CELL 19

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[8]	FPGA Fabric	
IN	NC	USBA_DATAI[5]	GPIOA_IN[8]	IN_A	F2H_GPIN[19]
OUT	SPISS_0_OUT[1]	DATAO[5]	GPIO_OUT[8]	IN_B	F2H_SCP[19]
OE	SPI_0_MASTER	~NDATAOE	GPIO_OE[8]	OUT_A	H2F_A[19]
				OUT_B	H2F_B[19]

16.4.7.10 IOMUX CELL 20

I/O pad name: SPI0_SS2_USBA_DATA6_MGPIO9A

Table 555 • IOMUX CELL 20

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[9]	FPGA Fabric	
IN	NC	USBA_DATA[6]	GPIOA_IN[9]	IN_A	F2H_GPIN[20]
OUT	SPISS_0_OUT[2]	DATAO[6]	GPIO_OUT[9]	IN_B	F2H_SCP[20]
OE	SPI_0_MASTER	~NDATAOE	GPIO_OE[9]	OUT_A	H2F_A[20]
				OUT_B	H2F_B[20]

16.4.7.11 IOMUX CELL 21

I/O pad name: SPI0_SS3_USBA_DATA7_MGPIO10A

Table 556 • IOMUX CELL 21

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[10]	FPGA Fabric	
IN	NC	USBA_DATA[7]	GPIOA_IN[10]	IN_A	F2H_GPIN[21]
OUT	SPISS_0_OUT[3]	DATAO[7]	GPIO_OUT[10]	IN_B	F2H_SCP[21]
OE	SPI_0_MASTER	~NDATAOE	GPIO_OE[10]	OUT_A	H2F_A[21]
				OUT_B	H2F_B[21]

16.4.7.12 IOMUX CELL 22

I/O pad name: SPI0_SS4_MGPIO19A

Table 557 • IOMUX CELL 22

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[19]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[19]	IN_A	GND
OUT	SPISS_0_OUT[4]	GND	GPIO_OUT[19]	IN_B	GND
OE	SPI_0_MASTER	GND	GPIO_OE[19]	OUT_A	H2F_A[22]
				OUT_B	NC

16.4.7.13 IOMUX CELL 23

I/O pad name: SPI0_SS5_MGPIO20A

Table 558 • IOMUX CELL 23

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[20]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[20]	IN_A	GND
OUT	SPISS_0_OUT[5]	GND	GPIO_OUT[20]	IN_B	GND
OE	SPI_0_MASTER	GND	GPIO_OE[20]	OUT_A	H2F_A[23]
				OUT_B	NC

16.4.7.14 IO MUX CELL 24

I/O pad name: SPI0_SS6_MGPIO21A

Table 559 • IOMUX CELL 24

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[21]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[21]	IN_A	GND
OUT	SPISS_0_OUT[6]	GND	GPIO_OUT[21]	IN_B	GND
OE	SPI_0_MASTER	GND	GPIO_OE[21]	OUT_A	H2F_A[24]
				OUT_B	NC

16.4.7.15 IOMUX CELL 25

I/O pad name: SPI0_SS7_MGPIO22A

Table 560 • IOMUX CELL 25

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[22]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[22]	IN_A	GND
OUT	SPISS_0_OUT[7]	GND	GPIO_OUT[22]	IN_B	GND
OE	SPI_0_MASTER	GND	GPIO_OE[22]	OUT_A	H2F_A[25]
				OUT_B	NC

16.4.7.16 IOMUX CELL 26

I/O pad name: SPI1_SDI_MGPIO11A

Table 561 • IOMUX CELL 26

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[11]	FPGA Fabric	
IN	SPISDI_1_IN	NC	GPIOA_IN[11]	IN_A	F2H_GPIN[26]
OUT	GND	GND	GPIO_OUT[11]	IN_B	F2H_SCP[26]
OE	GND	GND	GPIO_OE[11]	OUT_A	H2F_A[26]
				OUT_B	H2F_B[26]

16.4.7.17 IOMUX CELL 27

I/O pad name: SPI1_SDO_MGPIO12A

Table 562 • IOMUX CELL 27

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[12]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[12]	IN_A	F2H_GPIN[27]
OUT	SPI_SDO_1_OUT	GND	GPIO_OUT[12]	IN_B	F2H_SCP[27]
OE	SPI_OEN_0	GND	GPIO_OE[12]	OUT_A	H2F_A[27]
				OUT_B	H2F_B[27]

16.4.7.18 IOMUX CELL 28

I/O pad name: SPI1_SS0_MGPIO13A

Table 563 • IOMUX CELL 28

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[13]	FPGA Fabric	
IN	SPISSI_1_IN	NC	GPIOA_IN[13]	IN_A	F2H_GPIN[28]
OUT	SPISS_1_OUT[0]	GND	GPIO_OUT[13]	IN_B	F2H_SCP[28]
OE	SPI_1_MASTER	GND	GPIO_OE[13]	OUT_A	H2F_A[28]
				OUT_B	H2F_B[28]

16.4.7.19 IOMUX CELL 29

I/O pad name: SPI1_SS1_MGPIO14A

Table 564 • IOMUX CELL 29

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[14]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[14]	IN_A	F2H_GPIN[29]
OUT	SPISS_1_OUT[1]	GND	GPIO_OUT[14]	IN_B	F2H_SCP[29]
OE	SPI_1_MASTER	GND	GPIO_OE[14]	OUT_A	H2F_A[29]
				OUT_B	H2F_B[29]

16.4.7.20 IOMUX CELL 30

I/O pad name: SPI1_SS2_MGPIO15A

Table 565 • IOMUX CELL 30

I/O Pad Ports	IOMUXCELL Interface				
	SPI_0	USB Controller	GPIO[15]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[15]	IN_A	F2H_GPIN[30]
OUT	SPISS_1_OUT[2]	GND	GPIO_OUT[15]	IN_B	F2H_SCP[30]
OE	SPI_1_MASTER	GND	GPIO_OE[15]	OUT_A	H2F_A[30]
				OUT_B	H2F_B[30]

16.4.7.21 IOMUX CELL 31

I/O pad name: SPI1_SS3_MGPIO16A

Table 566 • IOMUX CELL 31

I/O Pad Ports	IOMUXCELL Interface				
	SPI_1	USB Controller	GPIO[16]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[16]	IN_A	F2H_GPIN[31]
OUT	SPISS_1_OUT[3]	GND	GPIO_OUT[16]	IN_B	F2H_SCP[31]
OE	SPI_1_MASTER	GND	GPIO_OE[16]	OUT_A	H2F_A[31]
				OUT_B	H2F_B[31]

16.4.7.22 IOMUX CELL 32

I/O pad name: SPI1_SS4_MGPIO17A

Table 567 • IOMUX CELL 32

I/O Pad Ports	IOMUXCELL Interface				
	SPI_1	USB Controller	GPIO[17]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[17]	IN_A	GND
OUT	SPISS_1_OUT[4]	GND	GPIO_OUT[17]	IN_B	GND
OE	SPI_1_MASTER	GND	GPIO_OE[17]	OUT_A	H2F_A[32]
				OUT_B	NC

16.4.7.23 IOMUX CELL 33

I/O pad name: SPI1_SS5_MGPIO18A

Table 568 • IOMUX CELL 33

I/O Pad Ports	IOMUXCELL Interface				
	SPI_1	USB Controller	GPIO[18]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[18]	IN_A	GND
OUT	SPISS_1_OUT[5]	GND	GPIO_OUT[18]	IN_B	GND
OE	SPI_1_MASTER	GND	GPIO_OE[18]	OUT_A	H2F_A[33]
				OUT_B	NC

16.4.7.24 IOMUX CELL 34

I/O pad name: SPI1_SS6_MGPIO23A

Table 569 • IOMUX CELL 34

I/O Pad Ports	IOMUXCELL Interface				
	SPI_1	USB Controller	GPIO[23]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[23]	IN_A	GND
OUT	SPISS_1_OUT[6]	GND	GPIO_OUT[23]	IN_B	GND
OE	SPI_1_MASTER	GND	GPIO_OE[23]	OUT_A	H2F_A[34]
				OUT_B	NC

16.4.7.25 IOMUX CELL 35

I/O pad name: SPI1_SS7_MGPIO24A

Table 570 • IOMUX CELL 35

I/O Pad Ports	IOMUXCELL Interface				
	SPI_1	USB Controller	GPIO[24]	FPGA Fabric	
IN	NC	NC	GPIOA_IN[24]	IN_A	GND
OUT	SPISS_1_OUT[7]	GND	GPIO_OUT[24]	IN_B	GND
OE	SPI_1_MASTER	GND	GPIO_OE[24]	OUT_A	H2F_A[35]
				OUT_B	NC

16.4.7.26 IOMUX CELL 36

I/O pad name: MMUART1_RTS_MGPIO11B

Table 571 • IOMUX CELL 36

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[11]	FPGA Fabric	
IN	NC	NC	GPIOB_IN[11]	IN_A	F2H_GPIN[36]
OUT	RTS_1_OUT	GND	GPIO_OUT[11]	IN_B	F2H_SCP[36]
OE	VDD	GND	GPIO_OE[11]	OUT_A	H2F_A[36]
				OUT_B	H2F_B[36]

16.4.7.27 IOMUX CELL 37

I/O pad name: MMUART1_DTR_MGPIO12B

Table 572 • IOMUX CELL 37

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[12]	FPGA Fabric	
IN	NC	NC	GPIOB_IN[12]	IN_A	GND
OUT	DTR_1_OUT	GND	GPIO_OUT[12]	IN_B	GND
OE	VDD	GND	GPIO_OE[12]	OUT_A	H2F_A[37]
				OUT_B	NC

16.4.7.28 IOMUX CELL 38

I/O pad name: MMUART1_CTS_MGPIO13B

Table 573 • IOMUX CELL 38

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[13]	FPGA Fabric	
IN	CTS_1_IN	NC	GPIOB_IN[13]	IN_A	GND
OUT	GND	GND	GPIO_OUT[13]	IN_B	GND
OE	VDD	GND	GPIO_OE[13]	OUT_A	H2F_A[38]
				OUT_B	NC

16.4.7.29 IOMUX CELL 39

I/O pad name: MMUART1_DSR_MGPIO14B

Table 574 • IOMUX CELL 39

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[14]	FPGA Fabric	
IN	DSR_1_IN	NC	GPIOB_IN[14]	IN_A	GND
OUT	GND	GND	GPIO_OUT[14]	IN_B	GND
OE	VDD	GND	GPIO_OE[14]	OUT_A	H2F_A[39]
				OUT_B	NC

16.4.7.30 IOMUX CELL 40

I/O pad name: MMUART1_RI_MGPIO15B

Table 575 • IOMUX CELL 40

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[15]	FPGA Fabric	
IN	RI_1_IN	NC	GPIOB_IN[15]	IN_A	GND
OUT	GND	GND	GPIO_OUT[15]	IN_B	F2H_SCP[40]
OE	VDD	GND	GPIO_OE[15]	OUT_A	NC
				OUT_B	NC

16.4.7.31 IOMUX CELL 41

I/O pad name: MMUART1_DCD_MGPIO16B

Table 576 • IOMUX CELL 41

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[16]	FPGA Fabric	
IN	DCD_1_IN	NC	GPIOB_IN[16]	IN_A	GND
OUT	GND	GND	GPIO_OUT[16]	IN_B	F2H_SCP[41]
OE	VDD	GND	GPIO_OE[16]	OUT_A	NC
				OUT_B	NC

16.4.7.32 IOMUX CELL 42

I/O pad name: MMUART1_TXD_USBC_DATA2__MGPIO24B

Table 577 • IOMUX CELL 42

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[24]	FPGA Fabric	
IN	TXD_1_IN	USBC_DATAI[2]	GPIOB_IN[24]	IN_A	F2H_GPIN[42]
OUT	TXD_1_OUT	DATAO[2]	GPIO_OUT[24]	IN_B	F2H_SCP[42]
OE	MMUART1_TE	~NDATAOE	GPIO_OE[24]	OUT_A	H2F_A[42]
				OUT_B	H2F_B[42]

16.4.7.33 IOMUX CELL 43

I/O pad name: MMUART1_SCK_USBC_DATA4__MGPIO25B

Table 578 • IOMUX CELL 43

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[25]	FPGA Fabric	
IN	MMUART1_SCK_IN	USBC_DATAI[4]	GPIOB_IN[25]	IN_A	F2H_GPIN[43]
OUT	MMUART1_BAUDOUTN	DATAO[4]	GPIO_OUT[25]	IN_B	F2H_SCP[43]
OE	MMUART1_E_MST_SCK	~NDATAOE	GPIO_OE[25]	OUT_A	H2F_A[43]
				OUT_B	H2F_B[43]

16.4.7.34 IOMUX CELL 44

I/O pad name: MMUART1_RXD_USBC_DATA3__MGPIO26B

Table 579 • IOMUX CELL 44

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_1	USB Controller	GPIO[26]	FPGA Fabric	
IN	RXD_1_IN	USBC_DATAI[3]	GPIOB_IN[26]	IN_A	F2H_GPIN[44]
OUT	GND	DATAO[3]	GPIO_OUT[26]	IN_B	F2H_SCP[44]
OE	GND	~NDATAOE	GPIO_OE[26]	OUT_A	H2F_A[44]
				OUT_B	H2F_B[44]

16.4.7.35 IOMUX CELL 45

I/O pad name: MMUART0_RTS_USBC_DATA5__MGPIO17B

Table 580 • IOMUX CELL 45

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[17]	FPGA Fabric	
IN	NC	USBC_DATAI[5]	GPIOB_IN[17]	IN_A	F2H_GPIN[45]
OUT	RTS_0OUT	DTAO[5]	GPIO_OUT[17]	IN_B	F2H_SCP[45]
OE	VDD	~NDATAOE	GPIO_OE[17]	OUT_A	H2F_A[45]
				OUT_B	H2F_B[45]

16.4.7.36 IOMUX CELL 46

I/O pad name: MMUART0_DTR_USBC_DATA6__MGPIO18B

Table 581 • IOMUX CELL 46

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[18]	FPGA Fabric	
IN	NC	USBC_DATAI[6]	GPIOB_IN[18]	IN_A	F2H_GPIN[46]
OUT	DTR_0_OUT	DATAO[6]	GPIO_OUT[18]	IN_B	F2H_SCP[46]
OE	VDD	~NDATAOE	GPIO_OE[18]	OUT_A	H2F_A[46]
				OUT_B	H2F_B[46]

16.4.7.37 IOMUX CELL 47

I/O pad name: MMUART0_CTS_USBC_DATA7_MGPIO19B

Table 582 • IOMUX CELL 47

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[19]	FPGA Fabric	
IN	CTS_0_IN	USBC_DATA[7]	GPIOB_IN[19]	IN_A	F2H_GPIN[47]
OUT	GND	DATAO[7]	GPIO_OUT[19]	IN_B	F2H_SCP[47]
OE	GND	~NDATAOE	GPIO_OE[19]	OUT_A	H2F_A[47]
				OUT_B	H2F_B[47]

16.4.7.38 IOMUX CELL 48

I/O pad name: MMUART0_DSR_MGPIO20B

Table 583 • IOMUX CELL 48

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[20]	FPGA Fabric	
IN	DSR_0_IN	NC	GPIOB_IN[20]	IN_A	F2H_GPIN[48]
OUT	GND	GND	GPIO_OUT[20]	IN_B	F2H_SCP[48]
OE	GND	GND	GPIO_OE[20]	OUT_A	H2F_A[48]
				OUT_B	H2F_B[48]

16.4.7.39 IOMUX CELL 49

I/O pad name: MMUART0_RI_MGPIO21B

Table 584 • IOMUX CELL 49

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[21]	FPGA Fabric	
IN	RI_0_IN	NC	GPIOB_IN[21]	IN_A	F2H_GPIN[49]
OUT	GND	GND	GPIO_OUT[21]	IN_B	F2H_SCP[49]
OE	GND	GND	GPIO_OE[21]	OUT_A	H2F_A[49]
				OUT_B	H2F_B[49]

16.4.7.40 IOMUX CELL 50

I/O pad name: MMUART0_DCD_MGPIO22B

Table 585 • IOMUX CELL 50

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[22]	FPGA Fabric	
IN	DCD_0_IN	NC	GPIOB_IN[22]	IN_A	F2H_GPIN[50]
OUT	GND	GND	GPIO_OUT[22]	IN_B	F2H_SCP[50]
OE	GND	GND	GPIO_OE[22]	OUT_A	H2F_A[50]
				OUT_B	H2F_B[50]

16.4.7.41 IOMUX CELL 51

I/O pad name: MMUART0_TXD_USBC_DIR_MGPIO27B

Table 586 • IOMUX CELL 51

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[27]	FPGA Fabric	
IN	TXD_0_IN	USBC_DIR	GPIOB_IN[27]	IN_A	F2H_GPIN[51]
OUT	TXD_0_OUT	GND	GPIO_OUT[27]	IN_B	F2H_SCP[51]
OE	MMUART0_TE	GND	GPIO_OE[27]	OUT_A	H2F_A[51]
				OUT_B	H2F_B[51]

16.4.7.42 IOMUX CELL 52

I/O pad name: MMUART0_RXD_USBC_STP_MGPIO28B

Table 587 • IOMUX CELL 52

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[28]	FPGA Fabric	
IN	RXD_0_IN	NC	GPIOB_IN[28]	IN_A	F2H_GPIN[52]
OUT	GND	STP	GPIO_OUT[28]	IN_B	F2H_SCP[52]
OE	GND	VDD	GPIO_OE[28]	OUT_A	H2F_A[52]
				OUT_B	H2F_B[52]

16.4.7.43 IOMUX CELL 53

I/O pad name: MMUART0_SCK_USBC_NXT_MGPIO29B

Table 588 • IOMUX CELL 53

I/O Pad Ports	IOMUXCELL Interface				
	MMUART_0	USB Controller	GPIO[29]	FPGA Fabric	
IN	MMUART0_SCK_IN	USBC_NXT	GPIOB_IN[29]	IN_A	F2H_GPIN[53]
OUT	MMUART0_BAUDOUTN	GND	GPIO_OUT[29]	IN_B	F2H_SCP[53]
OE	MMUART0_E_MST_SCK	GND	GPIO_OE[29]	OUT_A	H2F_A[53]
				OUT_B	H2F_B[53]

16.4.7.44 IOMUX CELL 54

I/O pad name: I2C0_SDA_USBC_DATA0_MGPIO30B

Table 589 • IOMUX CELL 54

I/O Pad Ports	IOMUXCELL Interface				
	I2C_0	USB Controller	GPIO[30]	FPGA Fabric	
IN	SDA_0_IN	USBC_DATAI[0]	GPIOB_IN[30]	IN_A	F2H_GPIN[54]
OUT	GND	DATAO[0]	GPIO_OUT[30]	IN_B	F2H_SCP[54]
OE	~SDA_0_OUT	~NDATAOE	GPIO_OE[30]	OUT_A	H2F_A[54]
				OUT_B	H2F_B[54]

16.4.7.45 IOMUX CELL 55

I/O pad name: I2C0_SCL_USBC_DATA1_MGPIO31B

Table 590 • IOMUX CELL 55

I/O Pad Ports	IOMUXCELL Interface				
	I2C_0	USB Controller	GPIO[31]	FPGA Fabric	
IN	SCL_0_IN	USBC_DATAI[1]	GPIOB_IN[31]	IN_A	F2H_GPIN[55]
OUT	GND	DATAO[1]	GPIO_OUT[31]	IN_B	F2H_SCP[55]
OE	~SCL_0_OUT	~NDATAOE	GPIO_OE[31]	OUT_A	H2F_A[55]
				OUT_B	H2F_B[55]

16.4.7.46 IOMUX CELL 56

I/O pad name: USBD_DATA7_MGPIO23B

Table 591 • IOMUX CELL 56

I/O Pad Signals	IOMUXCELL Interface				
	I2C_0	USB Controller	GPIO[23]	FPGA Fabric	
IN	NC	USBD_DATAI[7]	GPIOB_IN[23]	IN_A	GND
OUT	GND	DATAO[7]	GPIO_OUT[23]	IN_B	GND
OE	GND	~NDATAOE	GPIO_OE[23]	OUT_A	NC
				OUT_B	NC

17 Communication Block

The communication block (COMM_BLK) provides a bi-directional message passing facility between the Cortex-M3 processor and the system controller, similar to a mailbox communication channel.

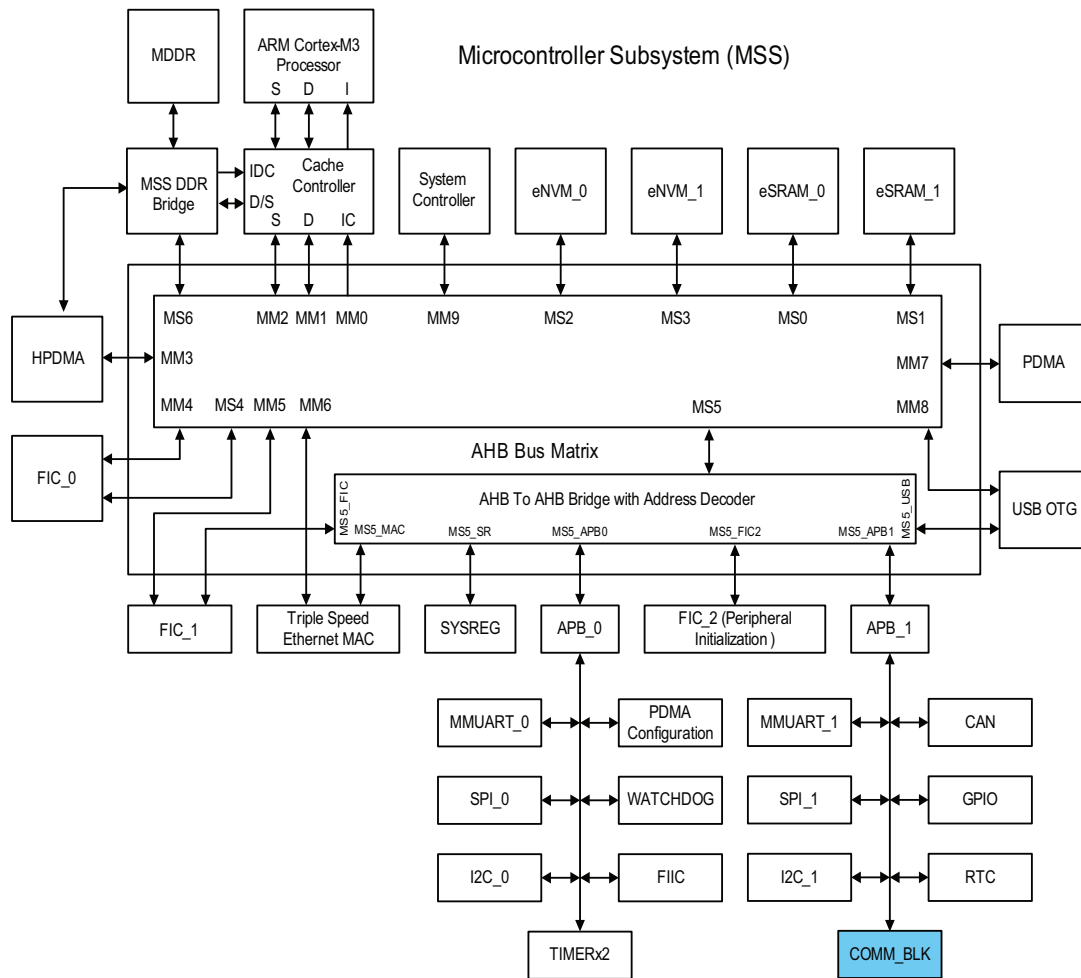
17.1 Features

The COMM_BLK peripheral includes the following features:

- Bi-directional byte-wide message path
- Supports serial data rate up to 50 Mbytes/sec.
- Asynchronous clock support
 - Data clock (50 MHz RC oscillator) is different from advanced peripheral bus (APB) clock
- 8 byte transmit FIFO
- 8 byte receive FIFO
- Flow control
 - RX to TX channels between microcontroller subsystem (MSS) COMM_BLK and system controller COMM_BLK
 - MSS COMM_BLK to peripheral direct memory access (PDMA) channel
- Frame and/or command marker
 - 9th bit used as frame start or command marker
 - Allows command and data sequences to be distinguished
 - Allows incomplete sequences to be detected
 - Separate command interrupt received with programmable match logic
- Allows WORD transfers into FIFO in a single APB cycle
- Interrupts
 - RX FIFO non-empty
 - TX FIFO non-full
 - TX overflow
 - RX Underflow

The following figure depicts the connectivity of COMM_BLK to the advanced high-performance bus (AHB) matrix.

Figure 250 • Interfacing of COMM_BLK with AHB Bus Matrix



17.2 Functional Description

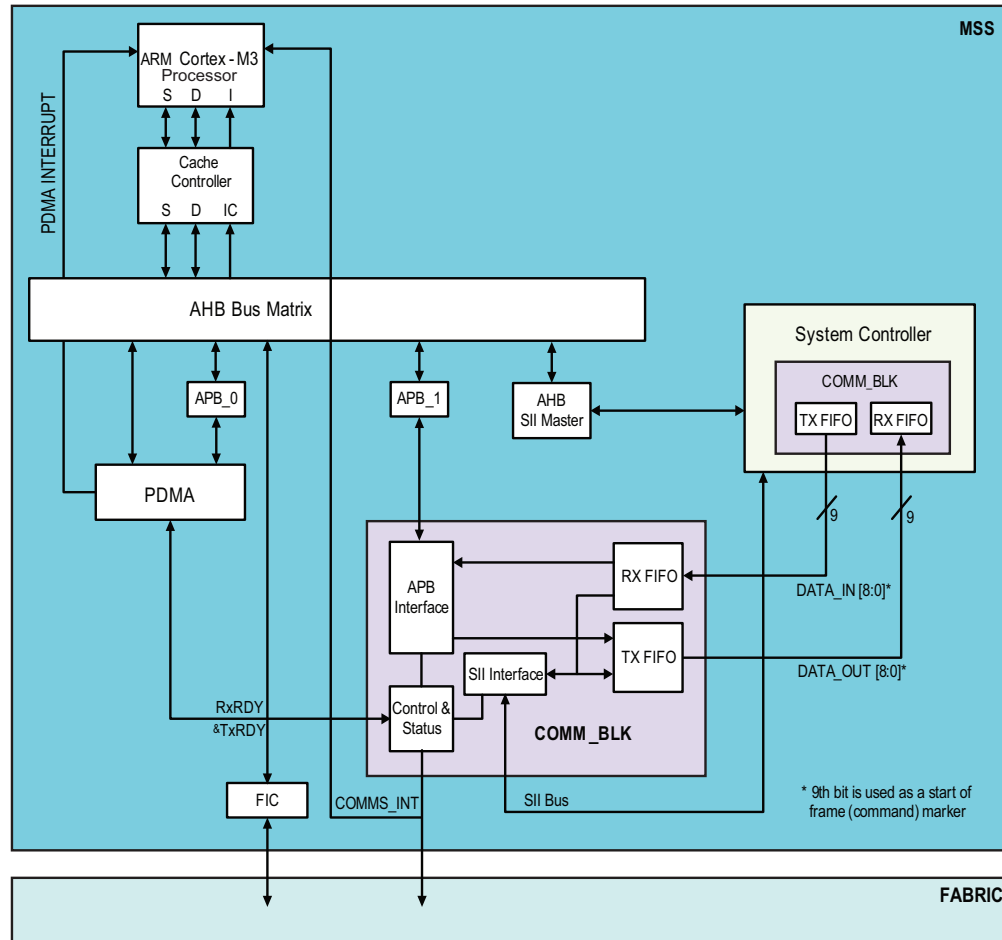
This section provides details of the COMM_BLK subsystem.

17.2.1 Architecture Overview

The COMM_BLK consists of an APB interface, 8 byte transmit FIFO, and an 8 byte receive FIFO. There is one COMM_BLK instantiated in the MSS and one in the system controller; each can communicate with the other. Whenever the Cortex-M3 processor writes a character into the COMM_BLK, it is transmitted to the receiving side of the COMM_BLK and an interrupt is asserted to the system controller.

In the other direction, the interrupt (COMM_BLK_INT) goes to both the Cortex-M3 processor and the FPGA fabric through the fabric interface interrupt controller (FIIC). This communication link is used as a message passing mailbox by firmware running on the Cortex-M3 processor and system controller. The following figure shows how COMM_BLKs are connected to create a communication channel between the Cortex-M3 processor and the system controller.

Figure 251 • Interfacing of COMM_BLK with System Controller



The COMM_BLK supports PDMA operation. The peripheral ready signals, RxRDY and TxRDY are directly connected to the PDMA, and are used for flow control between the MSS COMM_BLK and PDMA channel. Data from the COMM_BLK receive FIFO going to any MSS memory mapped locations, and the data from any MSS memory mapped locations going to the COMM_BLK transmit FIFO can be transferred without using the Cortex-M3 processor or the system controller. The PDMA supports DMA transfers from embedded nonvolatile memory (eNVM) to the COMM_BLK to facilitate the initialization of fabric SRAMs—Micro SRAM (uSRAM) and Large SRAM (LSRAM).

17.2.2 Frame/Command Marker

The COMM_BLK allows the data that is being transferred to be marked as a command or data byte. It is expected that a software protocol transfers packets of data between the COMM_BLK blocks. To allow the receiver to correctly identify the start of a packet, the COMM_BLK block uses a 9th bit (Bit 8 of DATA_IN and DATA_OUT as shown in [Figure 251 on page 594](#)).

When [FRAME_START8/FRAME_START32](#) register is written, the 9th bit is set. When [DATA8/DATA32](#) register is written, the 9th bit is not set.

The **STATUS** register bit 7 gives indication to the receiver whether the next byte that will be read out of the FIFO has the 9th bit set, and therefore indicating that it is the start of a packet.

This mechanism allows the receiver to verify that no bytes have been lost and stops it from accidentally interpreting data overruns as command. The RCVOKAY and TXTOKAY status bits must be checked in the [STATUS](#) register before reading and writing data or command.

17.2.3 Clocks

APB Interface, Control and Status block, and SII Interface are clocked by PCLK1 from the APB1 bus. RX FIFO and TX FIFO are clocked by data clock (50 MHz RC oscillator). PCLK is derived from the fabric aligned clock controller (FACC) output. Refer to the [UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide](#).

17.2.4 Resets

The COMM_BLK resets to zero on power-up and is held in reset until it is enabled. There is an option to reset the COMM_BLK by writing to the system register.

Specifically, this system register is [SOFT_RESET_CR](#) in the [System Register Block](#), page 670. The COMBLK_SOFTRESET control bit is encoded in bit location 15 as follows:

0: COMM_BLK reset released

1: COMM_BLK held in reset (reset value)

At power-up, the reset signal is asserted 1. This keeps the COMM_BLK peripheral in a reset state. If this bit is set to 0, the COMM_BLK peripheral is allowed to become active.

17.2.5 Interrupts

There is one interrupt signal from the COMM_BLK peripheral. The COMBLK_INTR/COMMS_INT signal is mapped to INTISR[19] in the Cortex-M3 processor nested vectored interrupt controller (NVIC) and also goes to the FPGA fabric through the FIIC. The interrupt in the COMM_BLK peripheral must be enabled by setting the appropriate bits in the interrupt enable register. Clear the appropriate bit in the [Interrupt Enable Register](#) when servicing the COMMS_INT to prevent a reassertion of the interrupt.

17.2.6 COMM_BLK Initialization

The COMM_BLK peripheral can be initialized by configuring the [COMM_BLK Control Register](#) and [SOFT_RESET_CR](#) system register. The initialization sequence is as follows:

1. Release the COMM_BLK from reset by using [SOFT_RESET_CR](#) system registry (refer to the [Resets](#), page 595 for further details)
2. Enable COMM_BLK by writing 1 to the ENABLE bit of [Control Register](#).
3. Disable the loopback by writing '0' to the LOOPBACK bit in the Control Register.

17.2.7 CoreSysServices Soft IP

COMM_BLK is used to call the following system services:

- Device and Design Information Services
- Flash*Freeze Service
- Cryptographic Services
- DPA-Resistant Key-Tree Services
- Non-Deterministic Random Bit Generator (NRBG) Services
- Zeroization Service
- Programming Service
- NVM Data Integrity Check Service

Microsemi provides CoreSysServices soft IP to access the system services implemented by the System Controller from FPGA fabric. The CoreSysServices soft IP provides a user interface for each of the system services and an advanced high-performance bus (AHB)-Lite master interface on the fabric interface controller (FIC) side. The core communicates with the COMM_BLK through one of the fabric interface controllers (FICs).

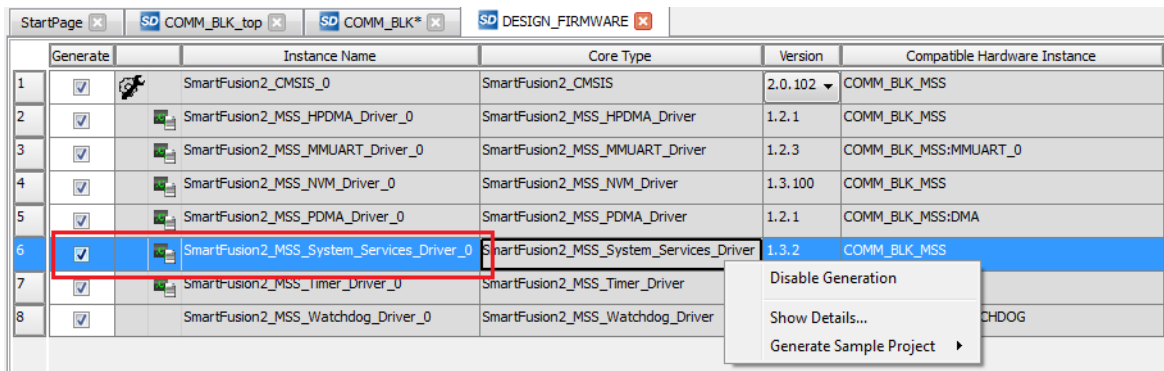
The CoreSysServices soft IP decodes the command received from the user logic and translates the user logic transactions to the AHB-Lite master transactions. For more information on CoreSysServices soft IP, refer to the **CoreSysServices Handbook** available in the *Libero SoC IP catalog*.

17.3 How to Use the Communication Block

17.3.1 COMM_BLK Configuration

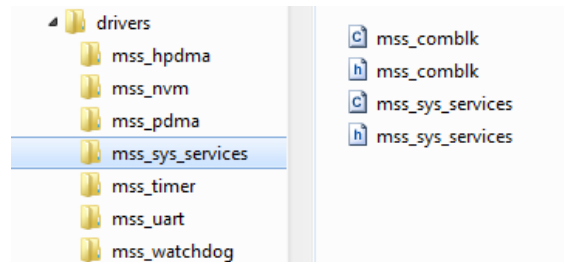
The COMM_BLK peripheral can be initialized and configured in the user application software by using application programming interfaces (APIs) available in COMM_BLK firmware driver. The COMM_BLK firmware driver is part of system services driver. The system services driver is enabled by default in firmware core configurator when the Libero SoC project is created. The following figure shows the system services driver in firmware core configurator.

Figure 252 • System Services Driver in Firmware Core Configurator



Once top-level component is generated (Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component** from the menu), the firmware and SoftConsole workspace are created in the project folder. The following figure shows the system services driver folder hierarchy.

Figure 253 • System Services Driver Folder Hierarchy



17.3.1.1 APIs

The following table lists the APIs for the COMM_BLK.

Table 592 • APIs for the COMM_BLK

Category	API	Description and Usage
Initialization	MSS_COMBLK_init	Initializes COMM_BLK
Write	MSS_COMBLK_send_cmd	Send command opcode and command parameters
	MSS_COMBLK_send_cmd_with_ptr	Send command opcode and command parameters pointer
	MSS_COMBLK_send_paged_cmd	Send command opcode and a page of data

Note: Microsemi recommends using system services driver provided in firmware core configurator for system service application development.

17.3.2 Use Model

The COMM_BLK is used to call the following system services:

- Device and design information services
- Flash*Freeze services
- Cryptographic services
- DPA-resistant key tree services
- Deterministic random bit generator services
- Zeroization service
- Programming services

Refer to the "System Services" chapter in the [UG0450: SmartFusion2 SoC and IGLOO2 FPGA System Controller User Guide](#) to know how to implement the system services.

Table 593 on page 597 through Table 600 on page 600 show the COMM_BLK Register Map.

17.4 COMM_BLK Configuration Registers

The COMM_BLK base address resides at 0x40016000 and extends to address 0x40016FFF in the Cortex-M3 processor memory map. The following table summarizes the control and status registers for the COMM_BLK.

Table 593 • COMM_BLK Register Map

Register Name	Address Offset	R/W	Reset Value	Description
CONTROL	0x00	R/W	0x00	Control Register
STATUS	0x04	R/W	0x00	Status Register
INT_ENABLE	0x08	R/W	0x00	Interrupt Enable
DATA8	0x10	R/W	0x00	Byte Data Register
DATA32	0x14	R/W	0x00000000	Word Data Register
FRAME_START8	0x18	R/W	0x00	Frame/Command Byte Register
FRAME_START32	0x1c	R/W	0x00000000	Frame/Command Word Register

17.5 COMM_BLK Register Interface Details

This section describes the COMM_BLK registers in detail.

17.5.1 Control Register

Table 594 • CONTROL

Bit Number	Name	R/W	Reset Value	Description
[7:6]	RESERVED	R	00	Reserved
5	LOOPBACK	R/W	1	After system reset the COMM_BLK is in Loopback mode. Set LOOPBACK bit to '0' to disable the loopback (Normal operation). It is used for factory test.
4	ENABLE	R/W	0	Configure the COMM_BLK interface. 0: Disables COMM_BLK 1: Enables COMM_BLK Enable COMM_BLK before writing to the FIFO and leave it enabled if it is being used.
3	SIZERX	R/W	0	Sets the number of bytes that each APB transfer reads from the RX FIFO. 0: 1 Byte 1: 4 Bytes (32-bits) This setting effects the behavior of the RxRDY signal and RCVOKAY flags. When set to 0 the flags indicate that a byte can be read and when set to 1 it indicates that a word can be read.
2	SIZETX	R/W	0	Sets the number of bytes that each APB transfer writes into the TX FIFO. 0: 1 Byte 1: 4 Bytes (32-bits) This setting effects the behavior of TxRDY signal and TXTOKAY. When set to 0 the flags indicate that a byte can be written and when set to 1 it indicates that a word can read be written.
1	FLUSHIN	R	0	Indicates FIFO flush status. 1 indicates flush process is in progress. 0 indicates that flush process is completed.
0	FLUSHOUT	R/W	0	Flush all FIFO's. Writing 1 to this bit starts the flush process. When the flush process is complete this bit returns to 0 automatically. The flush process takes several clock cycles to complete, depending on the various clock rates. Writing 0 has no effect.

17.5.2 Status Register

This register provides status information. R/W bits are cleared by writing 1. FIFO empty full flags will automatically clear as FIFO is full and empty.

Table 595 • STATUS

Bit Number	Name	R/W	Reset Value	Description
7	COMMAND	R	0	First byte queued in receive FIFO has the command marker set
6	SIIERROR	R/W	0	When an SII transfer (MSS to SII) is in progress, the start of frame marker is set on one or more of the bytes. Write 1 to clear
5	FLUSHRCVD	R/W	0	Indicates that a FLUSH has been received. Write 1 to clear

Table 595 • STATUS (continued)

4	SIIDONE	R/W	0	Indicated that the transfer to SII Bus is complete. Write 1 to clear
3	UNDERFLOW	R/W	0	Receive Overflow. Indicates that the receive FIFO was read when empty. Write 1 to clear
2	OVERFLOW	R/W	0	Transmit Overflow. Indicates that the Transmit FIFO was written when full. Write 1 to clear
1	RCVOKAY	R	0	RCV FIFO non empty. Indicates that 1 or 4 bytes may be read based on SIZERX.
0	TXTOKAY	R	1	TXT FIFO non full. Indicates that 1 or 4 bytes may be written depending on SIZETX.

Note: The system IP interface (SII) master connects the System Controller with all the internal elements. It is used to transfer data to and from the MSS memory space by the System Controller for System Services. It is also used for factory test but not available for customer.

17.5.3 Interrupt Enable Register

This register enables the COMMS_INT to be set whenever the corresponding bit is set in the [STATUS](#) register.

Table 596 • INT_ENABLE

Bit Number	Name	R/W	Reset Value	Description
[7:0]	ENABLE	R/W	0x00	Matches corresponding bit in status register 0: Disables Interrupt
				1: Enables Interrupt

17.5.4 Byte Data Register

This register writes a byte to the Transmit FIFO or reads a byte from the Receive FIFO. If the Transmit FIFO is full at the time of a write, an OVERFLOW will be set in the STATUS register. Similarly, if Receive FIFO is empty at the time of a read, an UNDERFLOW will be generated.

Table 597 • DATA8

Bit Number	Name	R/W	Reset Value	Description
[7:0]	DATA8	R/W	0x00	Write: Writes a byte to the MSS COMM_BLK Transmit FIFO Read: Reads a byte from the MSS COMM_BLK Receive FIFO

When the [DATA8](#) register is written, the command bit (Bit 8 on DATA) is set to 0, indicating that it is data. Writes to this register automatically set the SIZETX to 0 (1 byte), and reads set the SIZERX to 0 (1 byte).

17.5.5 Word Data Register

This register writes a word (32 bits) to the Transmit FIFO or reads a word from the Receive FIFO. If the Transmit FIFO has less than 4 spaces available at the time of a write, an OVERFLOW will be set in the [STATUS](#) register. Similarly, if Receive FIFO has less than 4 bytes available at the time of a read, an UNDERFLOW will be generated.

Table 598 • DATA32

Bit Number	Name	R/W	Reset Value	Description
[31:0]	DATA32	R/W	0x00000000	Write: Writes a word to the MSS COMM_BLK Transmit FIFO Read: Read a word from the MSS COMM_BLK Receive FIFO

The LSB is transferred on the DATA bus first. When the [DATA32](#) register is written, the command bit (Bit 8 on DATA) is set to 0, indicating that it is data. Writes to this register automatically set the SIZETX to 1 (4 bytes) and reads set the SIZERX to 1 (4 bytes).

17.5.6 Frame/Command Byte Register

This register writes a byte to the Transmit FIFO or reads a byte from the Receive FIFO. If the Transmit FIFO is full at the time of a write, an OVERFLOW will be set in the [STATUS](#) register. Similarly, if Receive FIFO is empty at the time of a read, an UNDERFLOW will be generated.

Table 599 • FRAME_START8

Bit Number	Name	R/W	Reset Value	Description
[7:0]	FRAME_START8	R/W	0x00	Write: Writes byte to the MSS COMM_BLK transmit FIFO Read: Read a byte from the MSS COMM_BLK receive FIFO

When the FRAME_START8 register is written, the command bit (Bit 8 on DATA) is set to 1, indicating the start of a frame, that is, the command byte. Writes to this register automatically set the SIZETX to 0 (1 byte), and reads set the SIZERX to 0 (1 byte). The [STATUS](#) register bit 7 indicates that this byte is a command.

17.5.7 Frame/Command Word Register

This register writes a word (32-bits) to the Transmit FIFO or reads a word from the Receive FIFO. If the Transmit FIFO has less than four spaces available at the time of a write, an OVERFLOW will be set in [STATUS](#) register. Similarly, if Receive FIFO has less than four bytes available at the time of a read, an UNDERFLOW will be generated.

Table 600 • FRAME_START32

Bit Number	Name	R/W	Reset Value	Description
[31:0]	FRAME_START32	R/W	0x00000000	Write: Writes a word to the MSS COMM_BLK transmit FIFO Read: Reads a word from the MSS COMM_BLK receive FIFO

The least significant bit (LSB) is transferred on the DATA bus first. When the [FRAME_START32](#) register is written, the command bit is set to 1, indicating the start of a frame, that is, command byte. The command bit (Bit 8 on DATA) will be set on the first byte for writes.

Writes to this register automatically sets the SIZETX to 1 (4 bytes) and reads set the SIZERX to 1 (4 bytes). The [STATUS](#) register bit 7 indicates that this word is a command.

18 RTC System

The SmartFusion2 real-time counter (RTC) system keeps track of seconds, minutes, hours, days, weeks, and years.

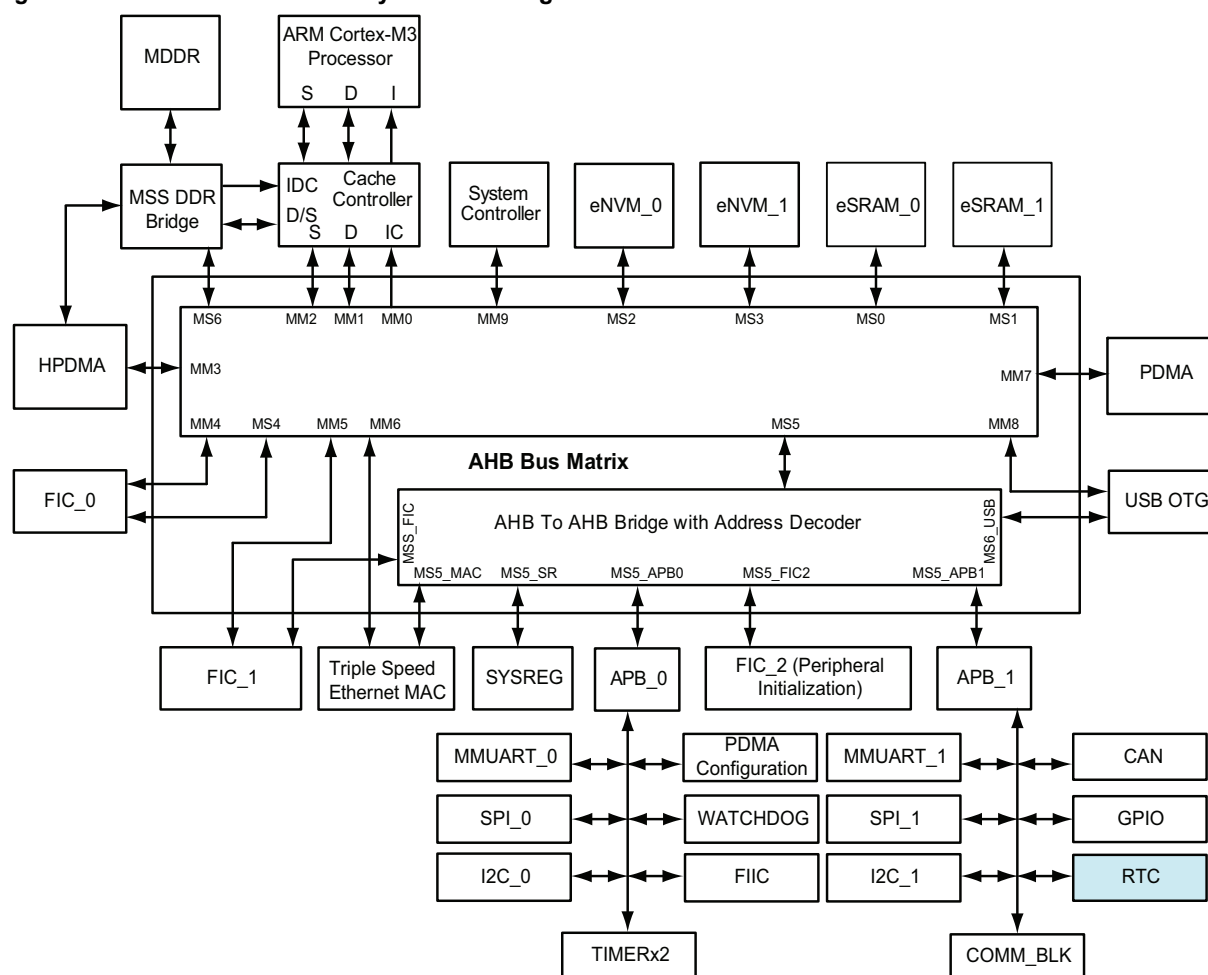
18.1 Features

It has two modes of operation:

- Real-time Calendar: Counts seconds, minutes, hours, days, week, months, and years
- Binary Counter: Consecutively counts from 0 to 243

As shown in the following figure, the RTC is connected to the AHB bus matrix through the APB_1 interface.

Figure 254 • Microcontroller Subsystem Showing RTC



18.2 Functional Description

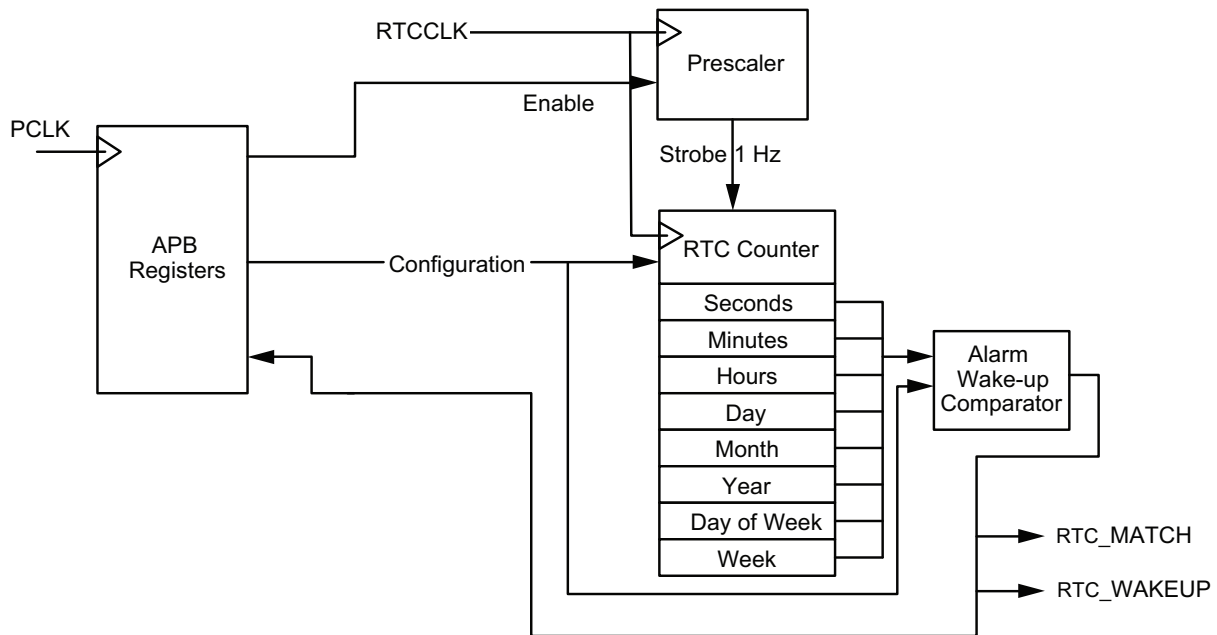
The following sections provide a detailed description of the RTC system.

18.2.1 Architecture Overview

This section describes the RTC architecture and its components which are as follows:

- Prescaler
- RTC Counter
- Alarm Wake-up Comparator

Figure 255 • RTC Block Diagram



18.2.1.1 Prescaler

The prescaler divides the input frequency to create a time-based strobe (typically 1 Hz) for the calendar counter. The Alarm and Compare Registers, in conjunction with the calendar counter, facilitate time-matched events.

To properly operate in Calendar mode, (Clock mode: 1), the 26-bit prescaler must be programmed to generate a 1 Hz strobe to the RTC. In Binary mode, (Clock mode: 0), the prescaler can be programmed as required in the application.

18.2.1.2 RTC Counter

The RTC counter keeps track of seconds, minutes, hours, days, weeks, and years when in Calendar mode, and for this purpose it requires a 43-bit counter. When counting in Binary mode, the 43-bit register is treated as a linear up counter.

The following table shows the details for Calendar mode and Binary mode.

Table 601 • Calendar Counter Description

Function	Number of Bits	Range		Reset Value	
		Calendar Mode	Binary Mode	Calendar Mode	Binary Mode
Second	6	0-59	0-63	0	0
Minute	6	0-59	0-63	0	0

Table 601 • Calendar Counter Description (continued)

Hour	5	0-23	0-31	0	0
Day	5	1-31 (auto adjust by month and year)	0-31	1	0
Month	4	1-12	0-15	1	0
Year	8	0-255 Year 2000 to 2255	0-255	0 (year 2000)	0
Weekday	3	1-7	0-7	1	0
Week	6	1-52	0-63	1	0

18.2.1.3 Alarm Wake-up Comparator

The RTC has two modes of operation, selectable through the clock_mode bit (Table 605, page 609).

In Calendar mode, the RTC counts seconds, minutes, hours, days, month, years, weekdays, and weeks. In Binary mode, the RTC consecutively counts from 0 all the way to 2^{43} . In both the modes, the alarm event generation logic simply compares the content of the Alarm register with that of the RTC; when they are equal, the RTC_MATCH output is asserted.

18.2.2 Port List

The following table lists the ports of the RTC and provides a short description for each.

Table 602 • RTC Interface Signals

Port	MSB	LSB	Dir	Description
PCLK			in	APB interface clock.
PRESETN			in	Processor reset
PADDR	6	0	in	APB address. Registers are word aligned so A1:0 is not used.
PSEL			in	APB select signal
PENABLE			in	APB enable
PWRITE			in	APB write signal
PWDATA	31	0	in	APB write data bus
PREADY			out	APB ready signal; is always asserted
PRDATA	31	0	out	APB read data bus
RTC_MATCH			out	RTC match output (active high) Synchronous to clk 32 k.
RTC_WAKEUP			out	RTC wake up interrupt (active high). Asserted Synchronous to clk 32 k, but deasserted on positive edge of PCLK.
CLKRTC			in	Clock input for RTC counters
PORST_B			in	Power-on reset. It clears/preset all flip-flops including the calendar/prescaler counters (active low).

18.2.2.1 Reset

The RTC is reset with the power-on reset (poreset_n) signal. Subsequent soft resets of the MSS do not reset the RTC. For further details, refer to the [Reset Controller](#), page 642.

18.2.2.2 Clocking

The RTC has two clock inputs:

- RTCCLK: This is used to clock the RTC.
- PCLK: This is used for the CPU interface.

The configuration bits in the mode registers should not be changed while the RTC is operational. The Alarm and Compare registers can be written by disabling the alarm (bit 2), as explained in the [Control Register](#), page 610.

18.2.2.2.1 Clock Source to RTCCLK

The MSS clock controller supplies three clock sources to RTCCLK:

- Crystal oscillator 32.767 KHz
- 1 MHz oscillator
- 50 MHz oscillator

The prescaler should be programmed to derive a 1 Hz signal. Therefore, for the 32.767 KHz clock, the prescaler should be programmed to 32768 (actual value is $N-1$, that is, 32767). Microsemi recommends that the lowest clock frequency source available is used because this reduces the power consumption; the MSS SYSREG provides the clock selection logic.

18.2.3 Details of Operation

The following sections describe the details of operation of the RTC:

- Day of the Week and Week Counter
- RTC_MATCH Status Bit and Output
- RTC_WAKEUP Status Bit
- RTC_WAKEUP Output

18.2.3.1 Day of the Week and Week Counter

The Calendar counter also keeps track of the days within the weeks, and the weeks in a year. These needs are to be set correctly before starting the RTC to match the day of the week and week for the current date and time. The day of the week counter increments from 1 to 7 and the week counter is incremented as the day of week goes from 7 to 1.

18.2.3.2 RTC_MATCH Status Bit and Output

The RTC_MATCH status bit and output is asserted whenever the Alarm system is enabled and a match occurs. In Calendar mode, it is asserted for a 1 second period while the alarm condition is valid. The output is synchronous to the rising edge of RTCCLK. The RTC_MATCH output signal can also be driven to the fabric.

18.2.3.3 RTC_WAKEUP Status Bit

The RTC_WAKEUP status bit is asserted whenever the Alarm system (refer to `alarm_enable` in [Table 607](#), page 610) is enabled and a match occurs. The bit stays set until cleared by writing to the control register clear wake-up bit.

18.2.3.4 RTC_WAKEUP Output

The RTC_WAKEUP output is asserted whenever the Alarm system is enabled (`alarm_enable`), the mode `wake_enable` ([Table 608](#), page 611) is set and a match occurs by the RTCCLK rising edge. The bit stays set until cleared by writing to the control register clear wakeup bit. The output de assertion is immediate and synchronous to the PCLK.

The RTC_WAKEUP output can be routed to the fabric through the fabric interface interrupt controller (FIIC) block in MSS and can be used by a soft microcontroller or a state machine implemented in the fabric. It can also be routed to the Cortex-M3 processor nested vectored interrupt controller (NVIC) or it can be routed to the system controller. The RTC_WAKEUP_CR in the SYSREG block provides masking for the RTC_WAKEUP interrupt to the fabric, the Cortex-M3 processor, and the system controller.

18.3 How to Use RTC

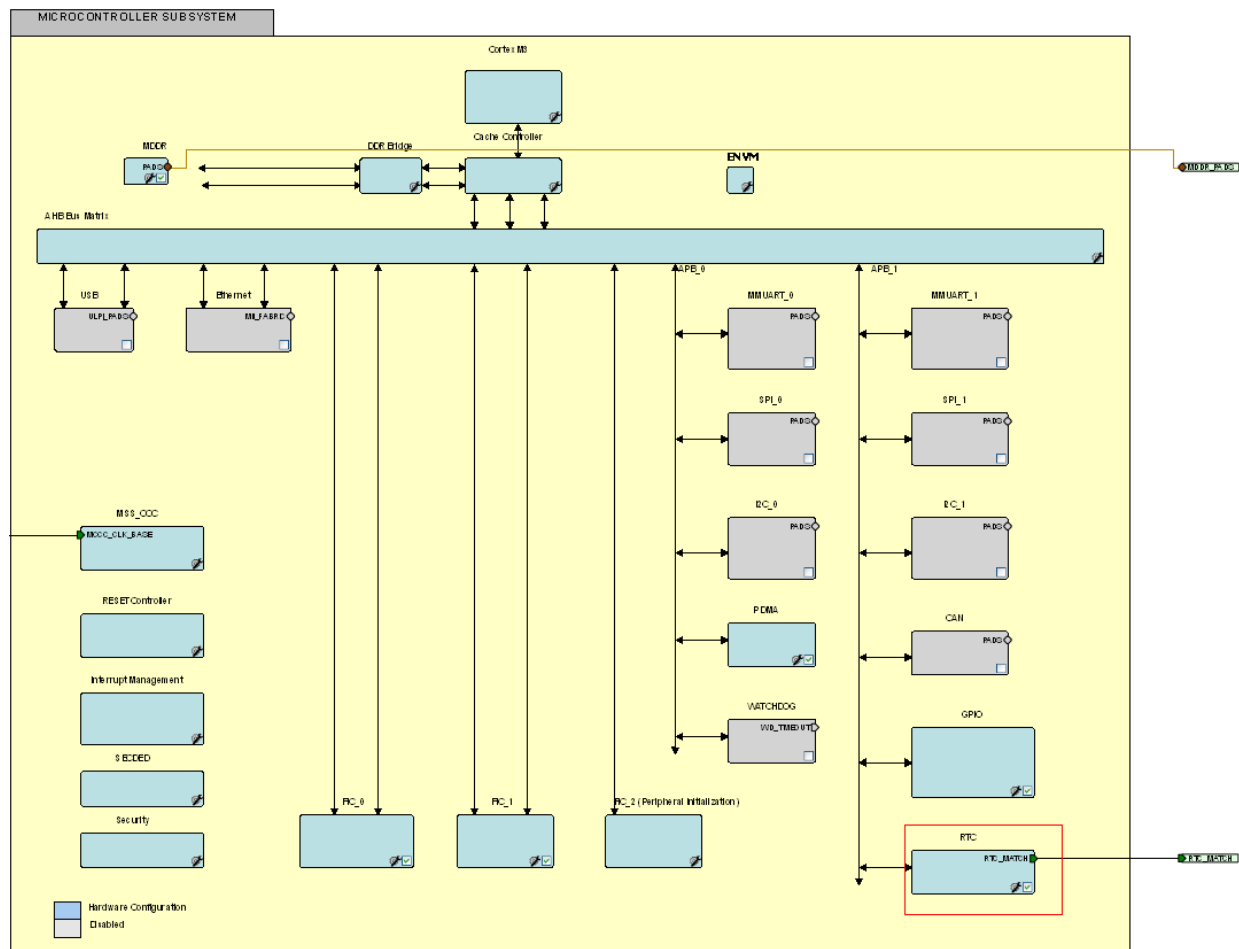
This section describes how to use the RTC in an application.

18.3.1 Design Flow

The following steps are used to enable the RTC in the application:

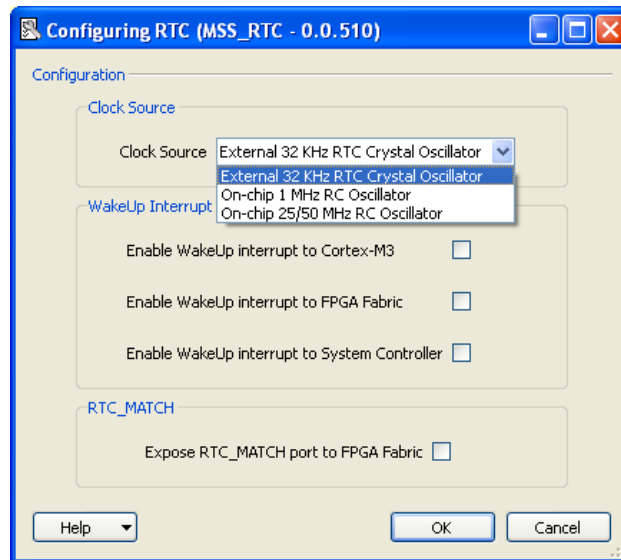
1. Enable RTC by using the MSS configurator in the application, as shown in the following figure.

Figure 256 • Enabling RTC in the Libero SOC Design MSS Configurator



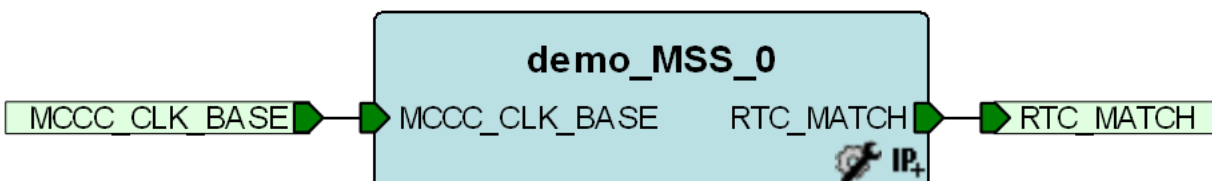
- Clicking RTC displays the RTC configuration window, as shown in the following figure.

Figure 257 • RTC Configuration Window

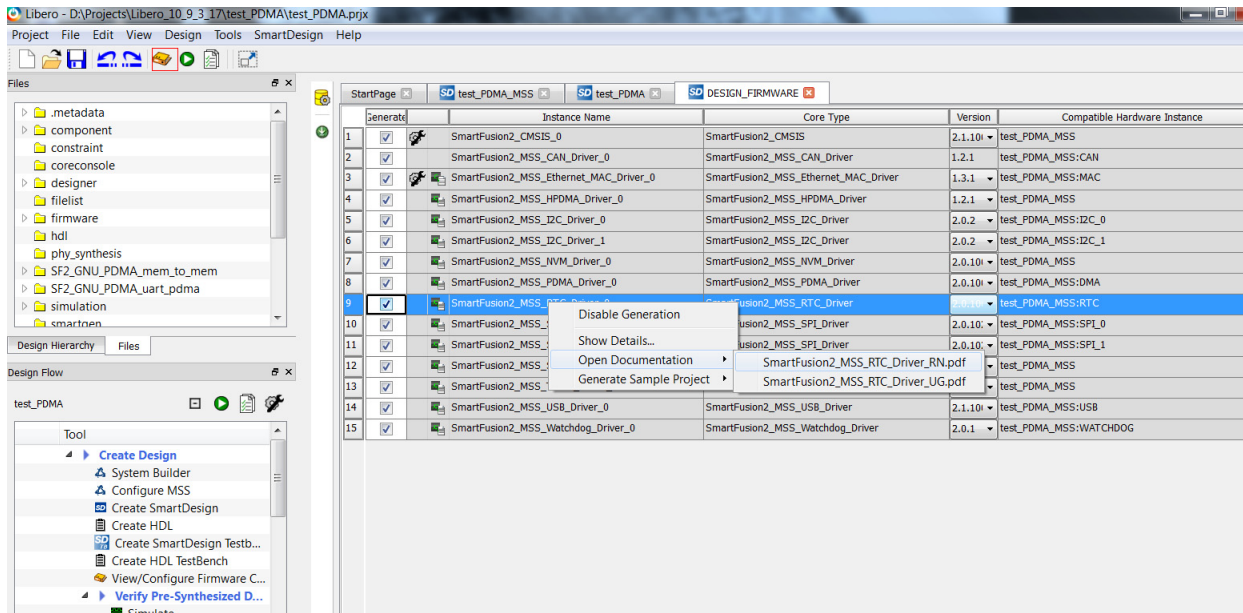


- Select the **Clock Source** that drives the RTC system (RTCCLK) as either External 32 KHz RTC crystal oscillator, or On-chip 1 MHz RC oscillator, or On-chip 25/50 MHz RC oscillator (50 MHz in 1.2 V part)
 - Select the **Wake-Up Interrupt**. The RTC_WAKEUP_CR in the SYSREG block provides masking for the RTC_WAKEUP interrupt to the FPGA fabric, the Cortex-M3 processor, and the system controller. The interrupt to be enabled can be selected using this configurator.
 - Select the **RTC_MATCH**. The RTC_MATCH status bit and output is asserted whenever the Alarm system is enabled and a match occurs. In Calendar mode, it is asserted for a 1 second period while the alarm condition is valid. The output is synchronous to the rising edge of the RTCCLK. The RTC_MATCH output signal can be exposed to drive the FPGA fabric. The RTC_MATCH signal is then available to be used in the design.
- The RTC signals in top level instance are shown in the following figure.

Figure 258 • RTC Signals



4. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component**. For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace is included in the project. Click the highlighted **Configure firmware** button as shown in the following figure to find the RTC drivers.

Figure 259 • RTC Driver User Guide


5. Click **Generate Bitstream** under **Program Design** to complete *.fdb file generation.
6. Double-click **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_rtc firmware driver. The firmware driver, mss_rtc (mss_rtc.c and mss_rtc.h) which provides a set of functions for controlling the RTC, can also be downloaded from the Microsemi firmware catalog. The following table lists the APIs for RTC. For more information on the APIs, refer to the **SmartFusion2_MSS_RTC_Driver_UG** (shown in the preceding figure).

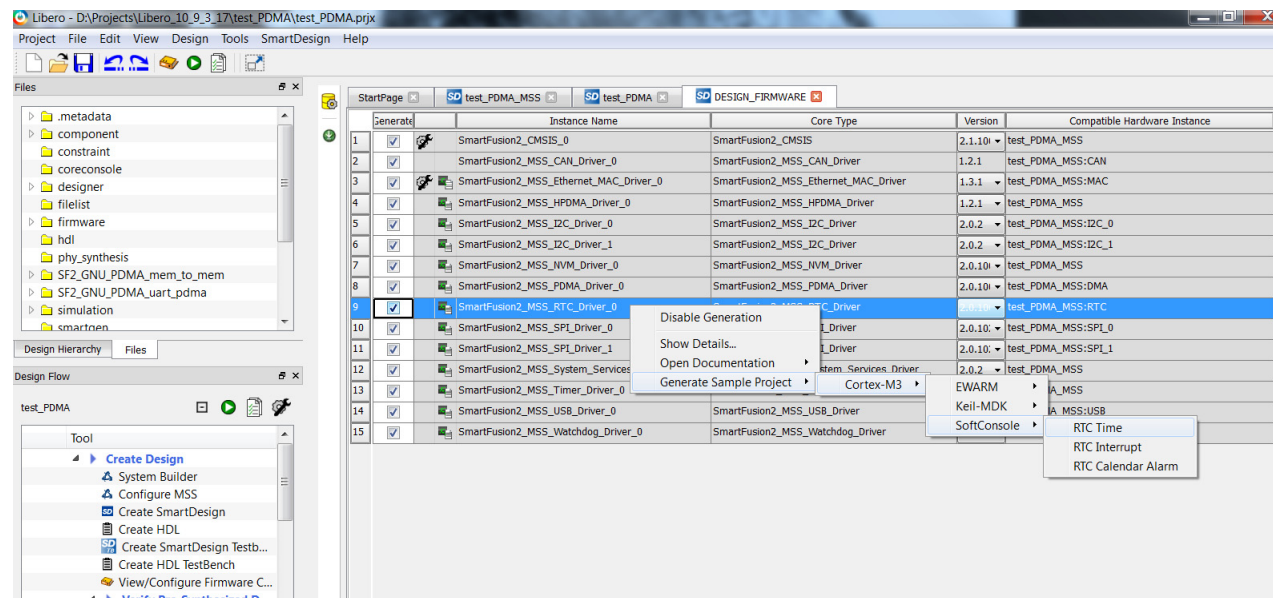
Table 603 • RTC APIs

Category	API	Description and Usage
Initialization	MSS_RTC_init()	Initializes RTC
Setting and reading the RTC counter current value	MSS_RTC_set_calendar_count()	Sets new calendar values to RTC counter
	MSS_RTC_set_binary_count()	Sets new binary values to RTC counter
	MSS_RTC_get_calendar_count()	Returns calendar count
	MSS_RTC_get_binary_count()	Returns binary counter value
Setting the RTC match and mask values	MSS_RTC_set_calendar_count_alarm()	Sets the RTC to generate an alarm when the time and date passed as parameter
	MSS_RTC_set_binary_count_alarm()	Sets the RTC to generate an alarm when the counter value passed as parameter
RTC counter increment detection	MSS_RTC_get_update_flag	Indicates if the RTC counter incremented since the last call to MSS_RTC_clear_update_flag()
	MSS_RTC_clear_update_flag	Clears the hardware flag set when the RTC counter increments

Table 603 • RTC APIs (continued)

Category	API	Description and Usage
Control of RTC	MSS_RTC_start()	Starts the RTC incrementing
	MSS_RTC_stop()	Stops the RTC from incrementing
	MSS_RTC_reset_counter()	Resets the calendar counters
Interrupt control	MSS_RTC_enable_irq()	Enables the RTC wakeup interrupt
	MSS_RTC_disable_irq()	Disables the RTC wake up interrupt
	MSS_RTC_clear_irq()	Clears pending RTC wakeup interrupt

7. For more information on RTC usage, sample projects that are available can be generated, as shown in the following figure.

Figure 260 • RTC Examples

18.3.2 RTC Use Model

Use the following steps to use RTC:

1. Configure the RTC from Libero SoC using the RTC configuration window.
2. Initialize the RTC driver using the MSS_RTC_init() function.
3. Use the MSS_RTC_configure() function to configure the RTC.
4. Set match value of the RTC using the MSS_RTC_set_rtc_match() function.
5. Use the MSS_RTC_enable_irq() function to enable the RTC match interrupt.
6. Call the MSS_RTC_start() function to increment RTC.
7. On reaching the match value, RTC generates the interrupt. Clear the interrupt.
8. Stop RTC increment using the MSS_RTC_stop() function

Note: The MSS RTC does not support full behavioral simulation models. Refer to *SmartFusion2 MSS BFM Simulation User Guide* for more information.

18.4 RTC Register Map

The Register Map for the RTC is shown in [Table 606](#), page 609.

Note: In this peripheral, all the register writes should be a WORD operation. A byte operation corrupts the other bits in the register.

18.4.1 Counter Bit Positions

The RTC counters support the following two counting modes:

- **Binary Mode:** In Binary mode a 43-bit counter is provided.
- **Calendar Mode:** The 43 bits are allocated as following:

Table 604 • Allocation of Bits in Calendar Mode

Counter	Counts	Size Bits	Bits in Counter	Reset Value
Seconds	0-59	6	[5:0]	0
Minutes	0-59	6	[11:6]	0
Hours	0-23	5	[16:12]	0
Day	1-31	5	[21:17]	1
Month	1-12	5	[25:22]	1
Year	0-255	8	[33:26]	0
Weekday	1-7	3	[36:34]	7
Week	1-52	6	[42:37]	1

- Weekday, 1: Sunday, 2: Monday 7: Saturday
- Reset date is Saturday 1 January 2000.
- The leap year calculations assume the year value where 0-255 is mapped to 2000 to 2255.

18.4.2 Register Bit Allocation

Depending on the CLOCK_MODE setting, the bit mapping is given in [Table 605](#), page 609.

Table 605 • Register Bit Allocation

Clock Mode	Date Time Alarm Compare Register	Bits	Description
0	Lower	[31:0]	Binary Count[31:0]
	Upper	[10:0]	Binary Count[42:32]
1	Lower	[7:0]	Seconds
		[15:8]	Minutes
		[23:16]	Hours
		[31:24]	Day
	Upper	[7:0]	Month
		[15:8]	Year
		[23:16]	Weekday
		[29:24]	Week

The following table shows the Register Map for RTC.

Table 606 • Register Map for RTC

Register Name	Address Offset	R/W	Reset Value	Description
Control	0x00	R/W	0	The control register is an 11-bit register that defines the operations of the RTC.
Mode	0x04	R/W	0	This register should only be written when the RTC is stopped—when the control register bit 0 reads as a '0'.

Table 606 • Register Map for RTC (continued)

Prescaler	0x08	R/W	0	The value of the prescaler can be written here. This register should only be written when the RTC is stopped—when the control register bit 0 reads as a '0'.
Alarm and Compare	0x0C-0x18	R/W	0	Sets and reads the alarm time.
Date and Time	0x20 -0x6C	R/W	0	Allows the individual bytes of the date and timer to be read; and registers can be written when the RTC is running.

18.4.3 Control Register

Table 607 • Control

Bit Number	Name	R/W	Reset Value	Description
10	Updated	R/W	0	This bit goes High every time the RTC updates the RTC value. It is cleared by writing '1' to the bit. It allows the CPU to know that the RTC value is stable and will not change for the next second.
9	Wakeup_set	W	0	Writing a '1' to this bit asserts the RTC_WAKEUP output and sets the Wakeup status bit.
8	Wakeup	R	0	This bit is set every time a match occurs. It is cleared by writing a '1'. Once cleared, it will not be set again until the next wakeup occurs. When WAKE_CONTINUE = 0, writing a '1' to this bit restarts the RTC counting after a wake-up occurs.
	Wakeup_clear	W		
7	Match	R	0	The RTC value matches the wakeup value. It indicates the value of the RTC_MATCH output. This is normally active for 1 second while the current time matches the alarm setting.
6	Download	W	0	Writing a '1' causes the current RTC value to be downloaded to the date/time upload registers initializing the upload value to the current time. This does not require any synchronization and takes place immediately.
5	Upload	R/W	0	When '1' is written, the date/time value loaded into the date/time registers is uploaded to the RTC and stays asserted until the internal synchronizers complete the upload.
4	Reset	W	0	When '1' is written, the date/time value is reset and uploaded to the RTC. This causes the upload bit to be set while the time is reset.
3	Alarm_off	W	0	When '1' is written, the Alarm is disabled.
	Alarm_enabled	R	0	When it is read, it indicates that the Alarm is enabled. It reads back '1' until the internal synchronization completes and the Alarm is fully disabled.
2	Alarm_on	W	0	When '1' is written, the Alarm is enabled.
	Alarm_enabled	R	0	When it is read, it indicates that the Alarm is enabled. It reads back '1' as soon as the bit is written.
1	Stop	W	0	When '1' is written, the RTC counter stops.
	Running	R	0	When it is read, it indicates that the RTC is running. It reads back as '1' until the internal synchronization completes and the RTC is fully stopped.

Table 607 • Control (continued)

Bit Number	Name	R/W	Reset Value	Description
0	Start	W	0	When '1' is written, the RTC starts.
	Running	R	0	When it is read, it indicates that the RTC is running. It reads back '1' as soon as the start bit is written.

18.4.4 Mode Register

This register should only be written when the RTC is stopped— when the control register bit 0 reads as a '0'.

Table 608 • Mode

Bit Number	Name	R/W	Reset Value	Description
4	Wake_reset_ps	R/W	0	When a wakeup occurs, resets the Prescaler.
3	Wake_continue	R/W	0	When a wakeup occurs, continues counting; otherwise the counters, including the Prescaler, stop until wake-up is cleared.
2	Wake_reset	R/W	0	When a wake-up occurs, resets the RTC.
1	Wake_enable	R/W	0	Enables the wakeup (interrupt) output.
0	Clock_mode	R/W	0	0: Binary counter 1: Calendar mode

18.4.5 Prescaler

This register should only be written when the RTC is stopped— when the control register bit 0 reads as a '0'.

Table 609 • Prescaler

Bit Number	Name	R/W	Reset Value	Description
[25:0]	Prescaler	R/W	0	The value by which the incoming clock is divided is set in the Prescaler. It should be set so as to achieve 1 Hz from the incoming RTC clock. Value: Clock Frequency – 1 Value must be greater than 1.

18.4.6 Alarm and Compare Registers

These registers may be written when the RTC is running, but the Alarm must be disabled; the control register bit 2 reads as a '0'.

Table 610 • Alarm and Compare

Address Offset	Register Name	Bit Numbers	Name	R/W	Reset Value	Description
0x0C	Alarm	[31:0]	Alarm Lower	R/W	0	Sets the alarm (wake-up) time on write and returns the alarm time on read.
0x10		[31:0]	Alarm Upper	R/W	0	Sets the alarm (wake-up) time on write and returns the alarm time on read.
0x14	Compare	[31:0]	Compare Lower	R/W	0	Sets the compare bits on the alarm time on write and returns the compare value on read. 0: Bit is ignored 1: Bit is compared
0x18		[31:0]	Compare Upper	R/W	0	Sets the compare bits on the alarm time on write and returns the compare value on read. 0: Bit is ignored 1: Bit is compared

Refer to [Table 605](#), page 609 for register bit allocation.

The Alarm setting must be set such that the ALARM*PRESCALER is greater than the 8 RTC clock cycles, assuming that the CPU writes to the control register to clear a wakeup condition within two RTC clock periods. Since the prescaler value should be set to achieve a 1 Hz pulse and the slowest RTCCLK source is 32 KHz, this requirement is always true.

18.4.7 Date and Time Registers

These registers may be written when the RTC is running. After writing these registers, the control register upload bit is used to upload the value coherently into the RTC counter.

Table 611 • Date and Time

Address Offset	Register Name	Bit Numbers	Name	R/W	Reset Value	Description
0x20	Date Time	[31:0]	Datetime Lower	R/W	0	Writes the data to be uploaded to the counter and returns the current time upon reading.
0x24		[31:0]	Datetime Upper	R/W	0	Writes the data to be uploaded to the counter and returns the upper time bits that are in alignment with the last read lower bits; that is, the value of the upper time bits as the lower ones are read.
0x30	Date/Time Synchronized Byte Mode	[5:0]	Seconds	R/W	0	Synchronized mode returns the date/time values at the point the second's register is read. Allows the individual byte of the date and timer to be read. The complete RTC data is read and stored internally when the second value is read, reads of minutes etc returns the value when seconds was read. For writes, all fields (0 × 30 - × 4C) must be written. The control register upload bit uploads data to the RTC. These registers are for use when clock_mode = 1.
0x34		[5:0]	Minutes	R/W	0	
0x38		[4:0]	Hours	R/W	0	
0x3C		[4:0]	Day	R/W	0	
0x40		[4:0]	Month	R/W	0	
0x44		[7:0]	Year	R/W	0	
0x48		[2:0]	Weekday	R/W	0	
0x4C		[5:0]	Week	R/W	0	

Table 611 • Date and Time (continued)

Address Offset	Register Name	Bit Numbers	Name	R/W	Reset Value	Description
0x50	Date/Time Direct Byte Mode	[5:0]	Seconds	R/W	0	Direct mode returns the date/time at the point that each of the the reads take place. Allows the individual byte of the date and timer to be read. Each read returns the current date/time value. It is possible that the date/time may increment between reads. For writing all fields (0 × 50 - × 6C) must be written. The control register upload bit uploads data to the RTC. These registers are for use when clock_mode = 1.
0x54		[5:0]	Minutes	R/W	0	
0x58		[4:0]	Hours	R/W	0	
0x5C		[4:0]	Day	R/W	0	
0x60		[4:0]	Month	R/W	0	
0x64		[7:0]	Year	R/W	0	
0x68		[2:0]	Weekday	R/W	0	
0x6C		[5:0]	Week	R/W	0	

Refer to [Table 605](#), page 609 for register bit allocation.

18.5 SYSREG Control Registers

The RTC_WAKEUP_CR in the SYSREG block provides masking for the RTC_WAKEUP interrupt to the fabric, the Cortex-M3 processor, and the system controller. Refer to the [System Register Block](#), page 670 for bit details.

Table 612 • The RTC_WAKEUP_CR in the SYSREG Block

Register Name	Register Type	Flash Write Protect	Reset Source	Description
RTC_WAKEUP_CR	RW-P	Register	sysreset_n	Provides masking for the RTC_WAKEUP interrupt to the fabric, the Cortex-M3 processor, and the system controller.

19 System Timer

The SmartFusion2 system timer (hereinafter referred as timer) consists of two programmable 32-bit decrementing counters that generate interrupts to the Cortex-M3 processor and FPGA fabric. The two 32-bit timers are identical. X is used as a placeholder for 1, 2, or 64 in register descriptions. It indicates Timer 1, Timer 2, or Timer 64.

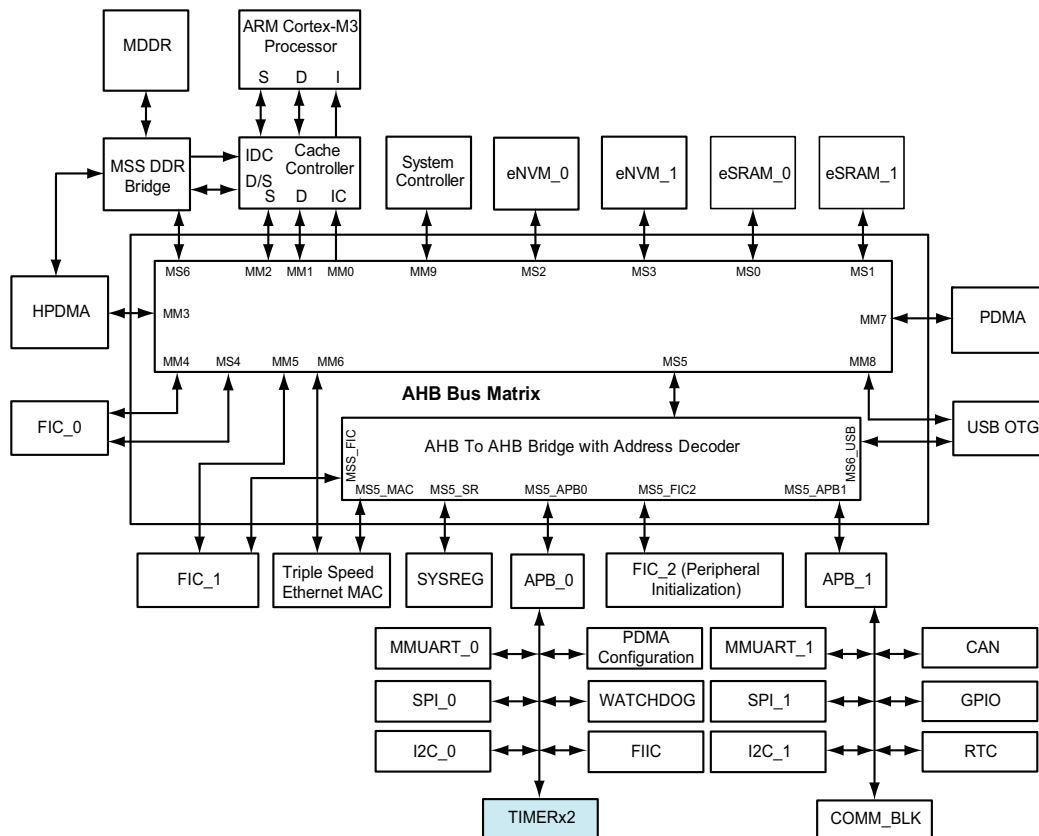
19.1 Features

The timer has the following features:

- Three count modes: One-shot and Periodic
- Decrementing 32-bit counters
- Two 32-bit timers can be concatenated to create a 64-bit timer.
- Option to enable or disable the interrupt requests when timer reaches zero.
- Controls to start, stop, and reset the timer

The following figure shows details of the MSS. Timer peripherals are connected to the AHB bus matrix through the APB_0 interface.

Figure 261 • MSS Showing Timer Peripherals



19.2 Functional Description

This section provides detailed description of the timer.

19.2.1 Architecture Overview

The timer is an APB_0 slave that provides two programmable, interrupt generating, 32-bit decrementing counters, as shown in the following figure.

Figure 262 • Timer Block Diagram

The Timer has an APB interface through which the Cortex-M3 processor can access various control and status registers to control and monitor the operation of the Timer. The Timer consists of two 32-bit decrementing counters. Counters generate the interrupts `TIMER1INT` and `TIMER2INT` on reaching zero. Refer to the [Timer Register Map](#), page 622 for more information on Timer registers.

19.2.2 Port List

The following table lists the Timer ports.

Table 613 • Timer Interface Signals

Name	Type	Width	Description
TIMER1INT	Output	1	Active high interrupt from counter 1. If enabled, this interrupt is asserted when counter 1 reaches zero. In 64-bit mode this interrupt line is asserted when the 64-bit counter reaches zero.
TIMER2INT	Output	1	Active high interrupt from counter 2. If enabled, this interrupt is asserted when counter 2 reaches zero.

19.2.2.1 Clocks

Timer is clocked by PCLK0 on the APB0 bus. PCLK is derived from the fabric alignment clock controller (FACC) output. Refer to the [UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide](#) for more information.

19.2.2.2 Resets

Timer resets to zero on power-up and is held in reset until enabled. Libero SoC software can reset the Timer by writing to bit 6 of `SOFT_RESET_CR` in the SYSREG block. Soft reset bit definitions are provided in the following table.

Table 614 • Soft Reset Bit Definitions for System Peripheral

Bit Number	Name	R/W	Reset Value	Description
6	TIMER_SOFTRESET	R/W	0x1	0: Releases the Timer from reset. 1: Keeps the Timer in reset.

19.2.2.3 Interrupts

There are two interrupt signals from the system timer block—TIMER1INT and TIMER2INT. The TIMER1INT signal is mapped to INTISR[14] and the TIMER2INT signal is mapped to INTISR[15] in the Cortex-M3 processor nested vectored interrupt controller (NVIC) controller. Both interrupt enable bits within the NVIC (INTISR[14] and INTISR[15]) correspond to bit locations 14 and 15. These interrupts are enabled by setting the appropriate TIMxINTEN bits in TIM1_CTRL, TIM2_CTRL, or TIM64_CTRL registers.

In 32-bit mode, the TIMxRIS bit in the respective interrupt service routine must be cleared to prevent a reassertion of the interrupt. Similarly, in 64-bit mode, TIM64RIS bit in the respective interrupt service routine must be cleared to prevent a reassertion of the interrupt.

19.2.3 Details of Operation

19.2.3.1 Timer Operating Modes

The Timers can be configured to operate in the following modes:

- Periodic Mode
- One-Shot Mode
- 64-Bit and 32-Bit Modes

The following sections explain the modes of operation.

19.2.3.1.1 Periodic Mode

In this mode, the counter generates the interrupts at constant intervals. On reaching zero, the counter is reloaded with a value held in a register and counting restarts.

Periodic mode is selected by setting the TIMxMODE bit in the [TIMx_CTRL](#) register to 0. In this mode, the counter continually counts down to zero when it is enabled. On reaching zero, an interrupt is generated and the counter is reloaded with the value stored in the [TIMx_LOADVAL](#) register. The counter then continues to count down towards zero, without waiting for the interrupt to be cleared. The interrupt remains asserted until cleared by the processor. If the counter reaches zero without clearing the previous interrupt, the counter behaves as if it has just timed out (reached zero). In effect, an interrupt is lost. It can continue indefinitely as long as the counter is enabled in Periodic mode and interrupts are not cleared.

Writing to the TIMxLOADVAL register at any time causes the counter to be loaded immediately with the value written, and, if enabled, it will continue counting down from the new value. If the [TIMx_BGLOADVAL](#) (background load value) register is written, the value written overwrites the TIMxLOADVAL register. The counter is not updated immediately with the new value. However, when the counter reaches zero, it is loaded with the new value contained in the TIMxLOADVAL register. By making use of the TIMxBGLOADVAL register, it is possible to continually generate interrupts with varying or alternating time intervals between interrupts without having any arbitrary variation in the lengths of the intervals due to (possibly) different cycle counts for servicing successive interrupts. For example, this approach is used to allow a processor to generate a waveform with a non-equal mark-space ratio on a general purpose output pin.

19.2.3.1.2 One-Shot Mode

The counter generates a single interrupt in this mode. On reaching zero, the counter halts until reprogrammed.

One-shot mode is selected by setting the TIMxMODE bit in the [TIMx_CTRL](#) register to 1. In this mode, the counter stops on reaching zero and a single interrupt is generated. When the counter is stopped in One-shot mode, it can be restarted by writing a non-zero value to the [TIMx_LOADVAL](#) register. Alternatively, the counter can be restarted by clearing the TIMxMODE bit. This causes the counter to be loaded with the value held in the [TIMx_LOADVAL](#) register and to begin operating in Periodic mode.

While the counter is counting down, it is possible to change the value of the TIMxMODE bit at any time without affecting the operation. For example, if the counter is decrementing in One-shot mode and the TIMxMODE bit is cleared before the counter reaches zero, the counter begins to operate in Periodic mode on reaching zero.

Writing to the TIMxBGLOADVAL register in One-shot mode has no real effect unless you intend to switch to Periodic mode when (or before) the next interrupt occurs. When in One-shot mode, the value written to TIMxBGLOADVAL is loaded into the TIMxLOADVAL register as normal but when the counter reaches zero, it generates a single interrupt and stops. Only a subsequent write to the TIMxLOADVAL register initiates another One-shot countdown sequence. However, if the counter is restarted by changing the Operating mode to Periodic (by clearing the TIMxMODE bit), the value written to the TIMxBGLOADVAL register is relevant because this is the start value (taken from the TIMxLOADVAL register) used to initialize the counter in Periodic mode.

19.2.3.1.3 64-Bit and 32-Bit Modes

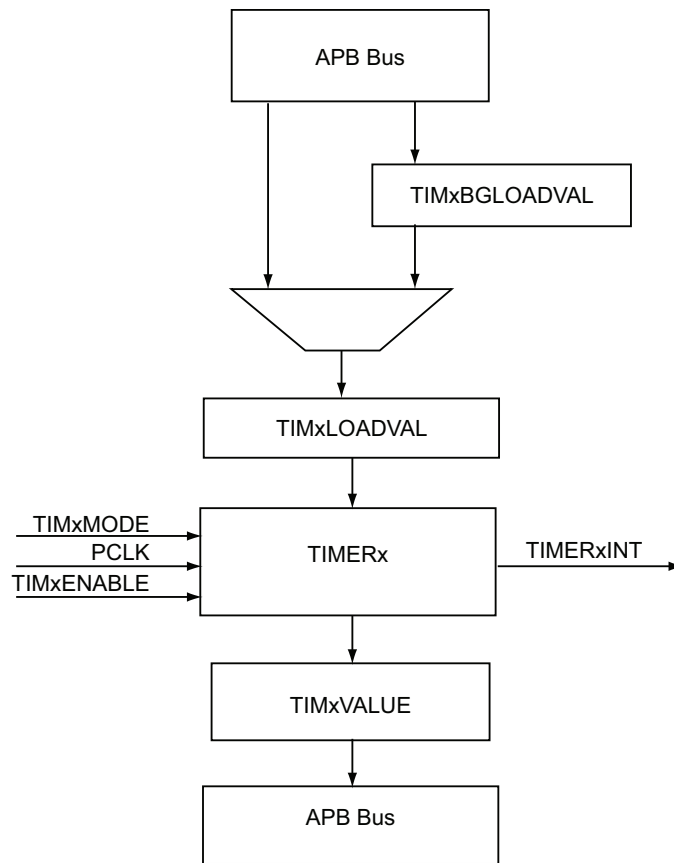
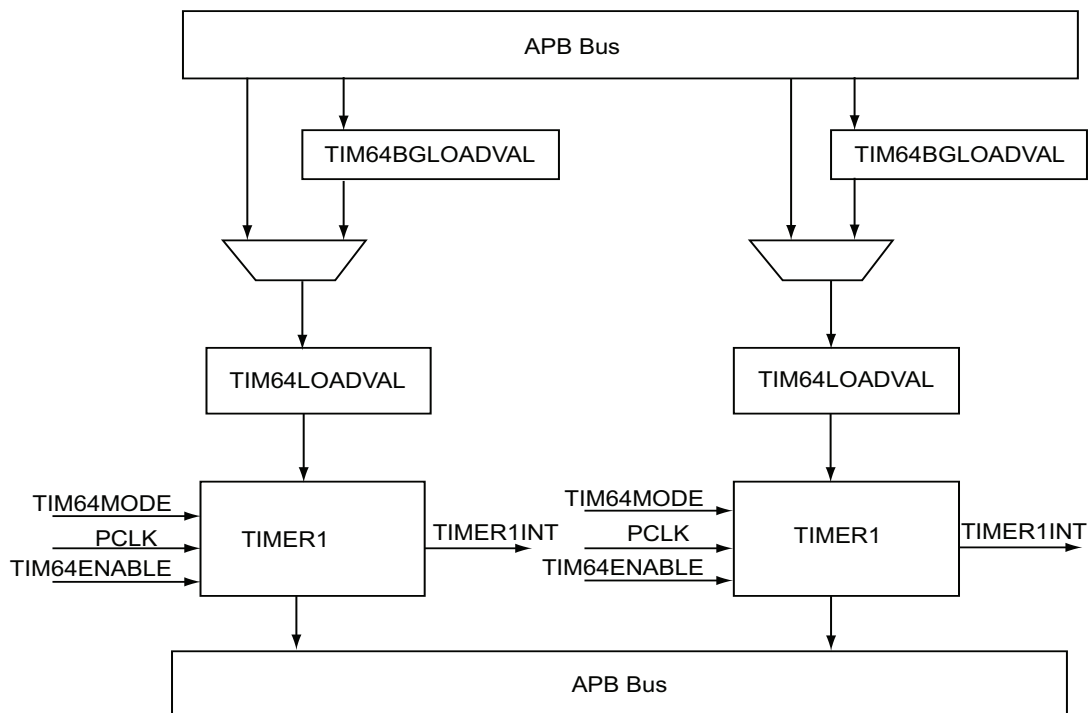
Timers 1 and 2 can be concatenated into a single 64-bit Timer that operates either in Periodic mode or in One-shot mode. [Figure 263](#), page 618 and [Figure 264](#), page 618 show block diagrams for 32-bit mode and 64-bit mode. Writing 1 to the TIM64_MODE register bit 0 sets the Timers in 64-bit mode. Whenever the TIM64_MODE bit changes state, the Timers are re-initialized to their default reset values. In 64-bit mode, writing to the 32-bit registers has no effect. Similarly, in 32-bit mode, writing to the 64-bit mode registers has no effect. Reading the 32-bit registers in 64-bit mode returns the default initialization values. Similarly, reading the 64-bit mode registers in 32-bit mode returns the default initialization values.

Timer 1 contains the lower 32-bit count of the 64-bit count value. Consequently, when updating or initializing the state of the counter, the upper 32 bits of the 64-bit counter must be written to first, followed by the lower 32 bits. It occurs as a two step process, first the upper 32 bits are written and then this value is stored in temporary register. When you write the lower 32 bits, the upper 32 bits are simultaneously written to the destination register. If a read is done on the target register before the write to the lower 32 bits is done, the functional value of the target register is returned (not the value of the temporary register).

While updating the background load value registers, ensure that TIM64_BGLOAD_VAL_U is followed by a write to TIM64_BGLOADVAL_L. When updating the load value registers, ensure TIM64_LOADVAL_U is followed by a write to TIM64_LOADVAL_L. When the lower 32-bit write occurs, the 64-bit counter is updated as one 64-bit value. There are temporary holding registers in the Timer block that are used to facilitate proper loading of the Timer in 64-bit mode. These registers are not readable.

In 64-bit mode, it is necessary to read 64-bit values as two 32-bit words. While reading TIM64_VAL_U and TIM64VAL_L, TIM64_VAL_L should be read first and then TIM64_VAL_U. When the TIM64_VAL_L (lower 32 bits of the 64-bit word) is read, the upper 32 bits are simultaneously read and put in a temporary register, which is returned while reading TIM64_VAL_U. This assures that the (changing) Timer values of the upper and lower words presented are sampled at the same time. The registers TIM64_BGLOADVAL_U, TIM64_BGLOADVAL_L, TIM64_LOADVAL_U, and TIM64_LOADVAL_L can be read in any order. Switching modes from 32-bit to 64-bit and vice versa requires changing the value in the mode register. Whenever this value is changed, the register values are restarted as the Timer is just reset.

Each 32-bit counter in the Timer is clocked with the PCLK input. With a PCLK frequency of 100 MHz, the maximum timeout period is approximately 42.9 seconds in 32-bit mode and 1.8×10^{11} seconds in 64-bit mode.

Figure 263 • Block Diagram 32-Bit Mode**Figure 264 • Block Diagram 64-Bit Mode**

19.3 How to Use Timer

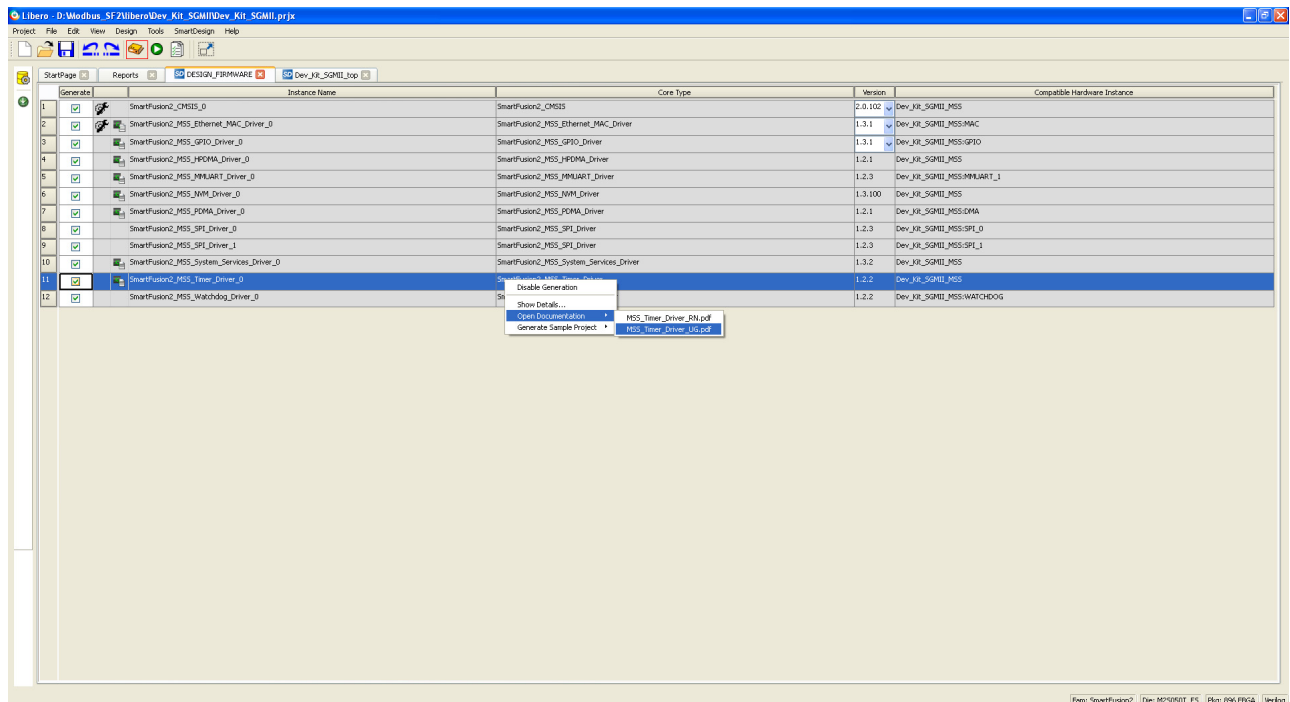
This section describes how to use the Timer in an application.

19.3.1 Design Flow

Follow the steps described below to use Timer in an application.

1. By default Timer module is enabled in the Libero project
2. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component**. For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace is included in the project. Click the highlighted **Configure firmware** as shown in the following figure to find the Timer drivers.

Figure 265 • Timer Driver User Guide



3. Click **Generate Bitstream** under **Program Design** to complete *.fdb file generation. Double-click **Export Firmware** under **Handoff Design** for **Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_timer firmware driver. The firmware driver, mss_timer (mss_timer.h) which provides a set of functions for controlling the Timer, can also be downloaded from the Microsemi firmware catalog. The following table lists the APIs for Timer.

For more information on the APIs, refer to the **SmartFusion2_MSS_Timer_Driver_UG** (shown in the preceding figure).

Table 615 • MSS Timer APIs

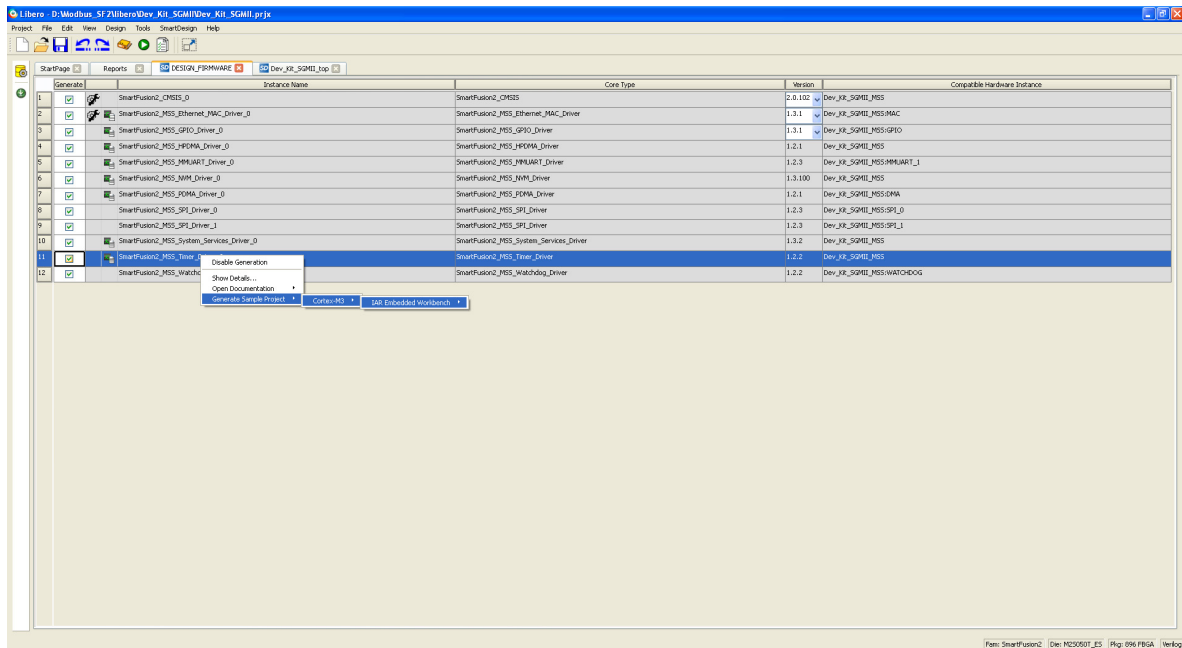
Category	API	Description and Usage
Initialization functions	MSS_TIM1_init()	Initializes Timer1
	MSS_TIM2_init()	Initializes Timer2
	MSS_TIM64_init()	Initializes 64-bit Timer

Table 615 • MSS Timer APIs (continued)

Category	API	Description and Usage
Control functions	MSS_TIM1_load_immediate()	Loads the value passed by the load_value parameter into the Timer1 down-counter.
	MSS_TIM1_load_background()	Specify the value that will be reloaded into the Timer1 down-counter the next time the counter reaches zero.
	MSS_TIM1_get_current_value()	Returns the current value of the Timer1 down-counter.
	MSS_TIM1_start()	The MSS_TIM1_start() function enables Timer1
	MSS_TIM1_stop()	The MSS_TIM1_stop() function disables Timer1
	MSS_TIM2_load_immediate()	Loads the value passed by the load_value parameter into the Timer2 down-counter
	MSS_TIM2_load_background()	Specify the value that will be reloaded into the Timer2 down-counter the next time the counter reaches zero.
	MSS_TIM2_get_current_value()	Returns the current value of the Timer 2 down-counter.
	MSS_TIM2_start()	Enables Timer 2
	MSS_TIM2_stop()	Disables Timer 2
	MSS_TIM64_load_immediate()	Loads the values passed by the load_value_u and load_value_l parameters into the 64-bit timer down-counter
	MSS_TIM64_load_background()	Specify the 64-bit value that will be reloaded into the 64-bit timer down-counter the next time the counter reaches zero
	MSS_TIM64_get_current_value()	Read the current value of the 64-bit timer down-counter.
	MSS_TIM64_start()	Enables the 64-bit timer
	MSS_TIM64_stop()	Disables the 64-bit timer
Interrupt control functions	MSS_TIM1_enable_irq()	Enable interrupt generation for Timer 1
	MSS_TIM1_disable_irq()	Disable interrupt generation for Timer 1
	MSS_TIM1_clear_irq()	Clear a pending interrupt from Timer 1
	MSS_TIM2_enable_irq()	Enable interrupt generation for Timer 2
	MSS_TIM2_disable_irq()	Disable interrupt generation for Timer 2
	MSS_TIM2_clear_irq()	Clear a pending interrupt from Timer 2
	MSS_TIM64_enable_irq()	Enable interrupt generation for the 64-bit timer
	MSS_TIM64_disable_irq()	Disable interrupt generation for the 64-bit timer
	MSS_TIM64_clear_irq()	Clear a pending interrupt from the 64-bit timer.

- For more information on Timer usage, the sample projects are available and can be generated, as shown in the following figure.

Figure 266 • Generating Sample Project



19.3.2 Timer Use Models

19.3.2.1 Use Model 1: 32-Bit Mode

To generate a 1 ms delay with a 100 MHz clock using 32-bit Timer 1 in Periodic mode:

- Initialize Timer for Periodic mode using `MSS_TIM1_init(MSS_TIMER_PERIODIC_MODE)`.
- Use `MSS_TIM1_load_immediate(load_value)` or `MSS_TIM1_load_background(load_value)` to load required time period. For a 1 ms delay load value calculation:
 - For 100 MHz, clock period is 0.01 μ s
 - Number of counts for a 1 ms delay is 1 ms / 0.01 μ s equal to 100,000 (186A0 in hexadecimal)
 - `MSS_TIM1_load_immediate(0x186A0)`
- Enable Timer 1 interrupt using `MSS_TIM1_enable_irq()`;
- Start the Timer using `MSS_TIM1_start()`;
- Whenever the counter reaches zero, it generates an interrupt.
- Clear the interrupt using `MSS_TIM1_clear_irq` and stop the Timer using `MSS_TIM1_stop()`.

Notes:

- `MSS_TIM1_clear_irq ()` function must be called as a part of implementation of the `Timer1_IRQHandler ()` Timer 1 interrupt service routine (ISR) in order to prevent the same interrupt event from retriggering a call to the ISR.
- The `MSS_TIM1_load_background()` function is used to specify the value that will be reloaded into the Timer 1 down-counter the next time the counter reaches zero. This function is typically used when Timer 1 is configured for Periodic mode operation to select or change the delay period between the interrupts generated by Timer 1.
- To calculate the load_value for different clock frequencies use formula `load_value = Hexadecimal value of (Timer Frequency \times Required delay)`

19.3.2.2 Use Model 2: 64-Bit Mode

To generate a 30 second delay with a 100 MHz clock using a 64-bit Timer in One-shot mode:

1. Initialize the Timer for One-shot mode using `MSS_TIM64_init(MSS_TIMER_ONE_SHOT_MODE)`.
2. Use `MSS_TIM64_load_immediate(load_value)` to load required time period
 For 30 seconds delay load value calculation:
 - For 100 MHz, clock period is 0.01 μ s
 - Number of counts for 30 second delay is $30/0.01 \mu$ s equal to 3000×10^6 (B2D05E00 in hexadecimal).
 - `MSS_TIM1_load_immediate(0x B2D05E00)`
3. Enable 64-bit Timer interrupt using `MSS_TIM64_enable_irq()`;
4. Start the Timer using `MSS_TIM64_start()`;
5. Whenever counter reaches zero, it generates an interrupt.
6. Clear the interrupt using `MSS_TIM64_clear_irq()` and stop the Timer `MSS_TIM64_stop()`.

19.4 Timer Register Map

The following table summarizes each of the Timer registers described in this document. The Timer base address resides at 0x40004000 and extends to address 0x40004FFF in the Cortex-M3 processor memory map.

Table 616 • Timer Register Map

Register Name	Address Offset	R/W	Reset Value	Description
TIM1_VAL (TIMx_VAL)	0x00	R	0	Current value of Timer1
TIM1_LOADVAL (TIMx_LOADVAL)	0x04	R/W	0	Load value for Timer1
TIM1_BGLOADVAL (TIMx_BGLOADVAL)	0x08	R/W	0	Background load value for Timer1
TIM1_CTRL (TIMx_CTRL)	0x0C	R/W	0	Control register for Timer1
TIM1_RIS (TIMx_RIS)	0x10	R/W	0	Timer 1 raw interrupt status
TIM1_MIS (TIMx_MIS)	0x14	R	0	Timer 1 masked interrupt status
TIM2_VAL (TIMx_VAL)	0x18	R	0	Current value of Timer2
TIM2_LOADVAL (TIMx_LOADVAL)	0x1C	R/W	0	Load value for Timer2
TIM2BGLOADVAL (TIMx_BGLOADVAL)	0x20	R/W	0	Background load value for Timer2
TIM2_CTRL (TIMx_CTRL)	0x24	R/W	0	Control register for Timer2
TIM2_RIS (TIMx_RIS)	0x28	R/W	0	Timer2 raw interrupt status
TIM2MIS (TIMx_MIS)	0x2C	R	0	Timer2 masked interrupt status
TIM64_VAL_U	0x30	R	0	Upper 32-bit word for 64-bit mode
TIM64_VAL_L	0x34	R	0	Lower 32-bit word for 64-bit mode
TIM64_LOADVAL_U (TIM64_VAL_U)	0x38	R/W	0	Upper 32-bit word for 64-bit mode immediate load
TIM64_LOADVAL_L	0x3C	R/W	0	Lower 32-bit word for 64-bit mode immediate load
TIM64_BGLOADVAL_U	0x40	R/W	0	Upper 32-bit word for background value for 64-bit mode
TIM64_BGLOADVAL_L	0x44	R/W	0	Lower 32-bit word for background value for 64-bit mode
TIM64_CTRL	0x48	R/W	0	Control register for 64-bit mode
TIM64_RIS	0x4C	R/W	0	Raw interrupt status for 64-bit mode

Table 616 • Timer Register Map (continued)

Register Name	Address Offset	R/W	Reset Value	Description
TIM64_MIS	0x50	R	0	Masked interrupt status for 64-bit mode
TIM64_MODE	0x54	R/W	0	Timer dual 32-bit or 64-bit

19.4.1 Timer x Value Register

Table 617 • TIMx_VAL

Bit Number	Name	R/W	Reset Value	Description
31:0	TIMx_VAL	R	0	This register holds the current value of the counter for Timer x. Reading this register when the timer is set to 64-bit mode returns the reset value. This is a read only register and writing in this register has no effect.

19.4.2 Timer x Load Value Register

Table 618 • TIMx_LOADVAL

Bit Number	Name	R/W	Reset Value	Description
31:0	TIMx_LOADVAL	R/W	0	This register holds the value to load into the counter for Timer x. When this register is written, the value written is loaded immediately into the counter, regardless of the mode Timer x is in (Periodic or One-shot). If Timer x is enabled, the counter starts decrementing from this value. When operating in Periodic mode, the value in this register is used to reload the counter when the counter decrements to zero. This register is overwritten, if the TIMx_BGLOADVAL register is written but the counter will not be updated with the new value in this case. In Periodic mode, TIMx_LOADVAL always stores the value, which is loaded into the counter. Writing or reading this register when the Timer is set to 64-bit mode has no effect. Reading this register in 64-bit mode returns reset value.

19.4.3 Timer x Background Load Value Register

Table 619 • TIMx_BGLOADVAL

Bit Number	Name	R/W	Reset Value	Description
31:0	TIMx_BGLOADVAL	R/W	0	When this register is written, the value written is loaded into the TIMx_LOADVAL register without updating the counter. This allows a new value to be loaded into the respective counter without interrupting the current count cycle. The counter is updated with the new value in TIMx_LOADVAL, when the counter decrements to 0. Writing this register when the Timer is set to 64-bit mode has no effect. Reading this register in 64-bit mode returns the reset value.

Timer x Control Register

Table 620 • TIMx_CTRL

Bit Number	Name	R/W	Reset Value	Description
31:3	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
2	TIMxINTEN	R/W	0	<p>Timer x interrupt enable. When the counter reaches zero, an interrupt is signaled to the Cortex-M3 processor NVIC, IRQ 14 for Timer 1, and IRQ 15 for Timer 2.</p> <p>0: Timer x interrupt disabled 1: Timer x interrupt enabled</p> <p>Writing this register when the Timer is set to 64-bit mode has no effect. Reading this register when the Timer is set to 64-bit mode returns the reset value.</p>
1	TIMxMODE	R/W	0	<p>Timer x mode.</p> <p>0: Timer x is in Periodic mode. If TIMxENABLE = 1 when the counter reaches zero, the counter is reloaded from the value in the TIMx_LOADVAL register and begins counting down immediately.</p> <p>1: Timer x is in One-shot mode. If TIMxENABLE = 1, when the counter reaches zero, the counter stops counting. To start the counter again, you must load TIMx_LOADVAL with a non-zero value or set the Timer to Periodic mode by clearing TIMxMODE to 0.</p> <p>Writing this register when the Timer is set to 64-bit mode has no effect. Reading this register when the Timer is set to 64-bit mode returns the reset value.</p>
0	TIMxENABLE	R/W	0	<p>Timer x enable.</p> <p>0: Timer x disabled 1: Timer x enabled</p> <p>Writing this register when the Timer is set to 64-bit mode has no effect. Reading this register when the Timer is set to 64-bit mode returns the reset value.</p>

19.4.4 Timer x Raw Interrupt Status Register

Table 621 • TIMx_RIS

Bit Number	Name	R/W	Reset Value	Description
31:1	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
0	TIMx_RIS	R/W	0	<p>Timer x raw interrupt status (RIS)</p> <p>0: Timer x has not reached zero 1: Timer x has reached zero at least once since this bit was last cleared (by a reset or by writing 1 to this bit).</p> <p>Writing a 1 to this bit clears the bit and the interrupt, writing a zero has no effect.</p> <p>Writing this register when the Timer is set to 64-bit mode has no effect. Reading this register when the Timer is set to 64-bit mode returns the reset value.</p>

19.4.5 Timer x Masked Interrupt Status Register

Table 622 • TIMx_MIS

Bit Number	Name	R/W	Reset Value	Description
31:1	Reserved	R	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
0	TIMx_MIS	R	0	Timer x masked interrupt status (MIS) This read only bit is a logical AND of the TIMxRIS and TIMxINTEN bits. The TIMxINT output from the Timer has the same value as this bit. Writing to this bit has no effect. Reading this register when the Timer is set to 64-bit mode returns the reset value.

19.4.6 Timer 64 Value Upper Register

Table 623 • TIM64_VAL_U

Bit Number	Name	R/W	Reset Value	Description
31:0	TIM64_VAL_U	R	0	This register holds the current value of the upper 32 bits of the 64-bit count value for the Timer. TIM64_VAL_L must be read before TIM64_VAL_U. This register is read only; writes have no effect. Reading this register when the Timer is set to 32-bit mode returns the reset value.

19.4.7 Timer 64 Value Lower Register

Table 624 • TIM64_VAL_L

Bit Number	Name	R/W	Reset Value	Description
31:0	TIM64_VAL_L	R	0	This register holds the current value of the lower 32 bits of the 64-bit count value for the Timer. This register is read only; writes have no effect. When reading from this register, the upper 32 bits of the 64-bit counter is stored into TIM64_VAL_U. To properly read the 64-bit counter value, you must read from this register first, then the TIM64_VAL_U. Reading this register when the Timer is set to 32-bit mode returns the reset value.

19.4.8 Timer 64 Load Value Upper Register

Table 625 • TIM64_LOADVAL_U

Bit Number	Name	R/W	Reset Value	Description
31:0	TIM64_LOADVAL_U	R/W	0	This register holds the upper 32-bit value to load into the Timer when in 64-bit mode. When this register is written, the value written is loaded immediately into a temporary register. The value in the temporary register is only written to the Timer when the lower 32-bit word TIM64_LOADVAL_L is written. Writing this register when the Timer is set to 32-bit mode has no effect. Reading this register when the Timer is set to 32-bit mode returns the reset value.

19.4.9 Timer 64 Load Value Lower Register

Table 626 • TIM64_LOADVAL_L

Bit Number	Name	R/W	Reset Value	Description
31:0	TIM64_LOADVAL_L	R/W	0	When this register is written, the value written is loaded immediately into the lower 32 bits of the 64-bit counter along with the value written in register TIM64_LOADVAL_U. This applies to both Periodic and One-shot mode. The value stored in this register is also used to reload the counter, when the count reaches zero and the counter is operating in the Periodic mode. This register is overwritten, if the TIM64BGLOADVAL register is written, but the counter will not be updated with the new value. The TIM64BGLOADVAL register is an internal register to the Timer, used to concatenate the two 32-bit values from TIM64_BGLOAD_VAL_U and TIM64_BGLOAD_VAL_L. If Periodic mode is selected, the values in the TIM64_LOADVAL_L and TIM64_LOADVAL_U are loaded into the counter when the counter decrements to zero. Writing this register when the Timer is set to 32-bit mode has no effect. Reading this register when the Timer is set to 32-bit mode returns the reset value.

19.4.10 Timer 64 Background Load Value Upper Register

Table 627 • TIM64_BGLOADVAL_U

Bit Number	Name	R/W	Reset Value	Description
31:0	TIM64_BGLOADVAL_U	R/W	0	This register holds the upper 32-bit background value to load into the Timer when in 64-bit mode. When this register is written, the value written is loaded into a temporary register without updating the background load value TIM64_BGLOADVAL_U. TIM64_BGLOADVAL_U is only updated when the lower 32-bit word is written. Reading this register returns the upper 32 bits of the current background register, which is to be loaded into the counter when the counter reaches zero. This value reflects the last write to TIM64_BGLOADVAL_U when the lower 32 bits are written. Writing this register when the Timer is set to 32-bit mode has no effect. Reading this register when the Timer is set to 32-bit mode returns the reset value.

19.4.11 Timer 64 Background Load Value Lower Register

Table 628 • TIM64_BGLOADVAL_L

Bit Number	Name	R/W	Reset Value	Description
31:0	TIM64_BGLOADVAL_L	R/W	0	Background load value for the lower 32 bits of 64-bit Timer. When this register is written, both the upper and lower words are written into an internal 64-bit TIM64LOADVAL register without updating the counter. The TIM64LOADVAL register is an internal register to the Timer used in 64-bit mode for concatenating the two 32-bit registers, TIM64_LOADVAL_U and TIM64_LOADVAL_L. The new 64-bit load value is loaded into the counter when the counter reaches zero. Writing this register when the Timer is set to 32-bit mode has no effect. Reading this register when the Timer is set to 32-bit mode returns the reset value.

19.4.12 Timer 64 Control Register

Table 629 • TIM64_CTRL

Bit Number	Name	R/W	Reset Value	Description
31:3	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
2	TIM64INTEN	R/W	0	Timer 64 interrupt enable. When the counter reaches zero, an interrupt is signaled to the Cortex-M3 processor NVIC. 0: Timer 64 interrupt disabled 1: Timer 64 interrupt enabled
1	TIM64MODE	R/W	0	Timer 64 mode. 0: Timer 64 in Periodic mode If TIM64ENABLE = 1 when the counter reaches zero, the counter is reloaded from the value in the TIM64_LOADVAL_U and TIM64_LOADVAL_L registers and starts the counting down. 1: Timer 64 in One-shot mode If TIM64ENABLE = 1 when the counter reaches zero, the counter stops counting. To restart the counter, load TIM64_LOADVAL_U and TIM64_LOADVAL_L with a non-zero value or set the Timer to Periodic mode by clearing TIM64MODE to 0.
0	TIM64ENABLE	R/W	0	Timer 64 enable. 0: Timer 64 disabled 1: Timer 64 enabled Writing this register when the Timer is set to 32-bit mode has no effect. Reading this register when the Timer is set to 32-bit mode returns the reset value.

19.4.13 Timer 64 Raw Interrupt Status Register

Table 630 • TIM64_RIS

Bit Number	Name	R/W	Reset Value	Description
31:1	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
0	TIM64_RIS	R/W	0	Raw interrupt status (RIS) for 64-bit Timer mode. 0: Timer 64 has not reached zero 1: Timer 64 has reached zero at least once since this bit was last cleared (by a reset or by writing 1 to this bit). Writing 1 to this bit clears the bit and the interrupt; writing a zero has no effect.

19.4.14 Timer 64 Masked Interrupt Status Register

Table 631 • TIM64_MIS

Bit Number	Name	R/W	Reset Value	Description
31:1	Reserved	R	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
0	TIM64_MIS	R	0	Timer 64 masked interrupt status. This read only bit is a logical AND of the TIM64RIS and TIM64INTEN bits. The TIMER64INT output from the Timer has the same value as this bit. Writing to this bit has no effect.

19.4.15 Timer 64 Mode Register

Table 632 • TIM64_MODE

Bit Number	Name	R/W	Reset Value	Description
31:1	Reserved	R/W	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
0	TIM64_MODE	R/W	0	Timer 64 mode. 0: Timer 64 disabled; two separate 32-bit Timers. 1: Timer 64 enabled; one 64-bit Timer. Changing the state of this bit has the effect of reinitializing the Timer register map to its default power-up state.

20 Watchdog Timer

The watchdog timer is an advanced peripheral bus (APB) slave that guards against the system crashes by requiring regular service by the Cortex-M3 processor or by a bus master in the field programmable gate array (FPGA) fabric.

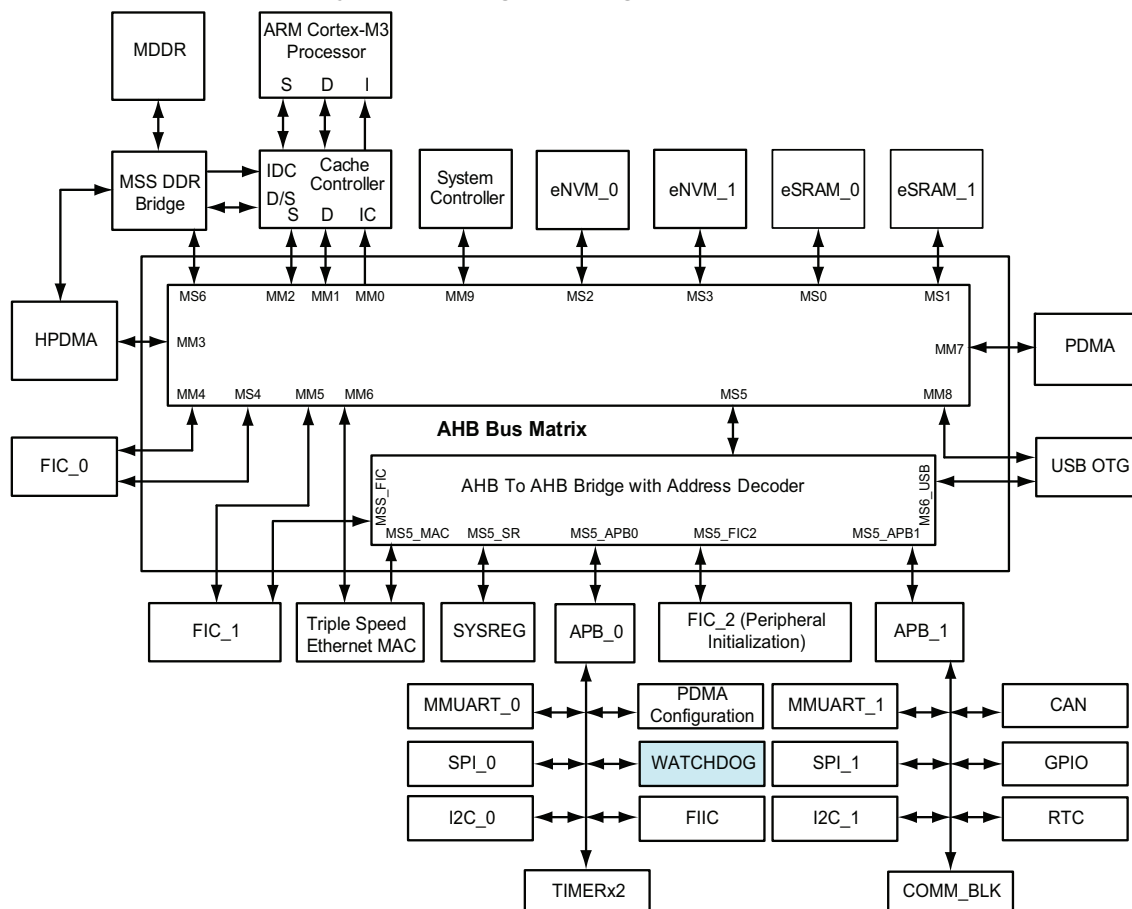
20.1 Features

The watchdog timer has following features:

- A 32-bit timer counts down from a preset value to zero, then performs one of the following user-configurable operations: If the counter is not refreshed, it will time out and either cause a system reset or generate an interrupt to the processor.
- The watchdog timer counter is halted when the Cortex-M3 processor enters the debug state.
- The watchdog timer can be configured to generate a wake up interrupt when the Cortex-M3 is in sleep mode.

As shown in the following figure, the watchdog timer is connected to the AHB bus matrix through the APB_0 interface.

Figure 267 • Microcontroller Subsystem Showing Watchdog Timer



20.2 Functional Description

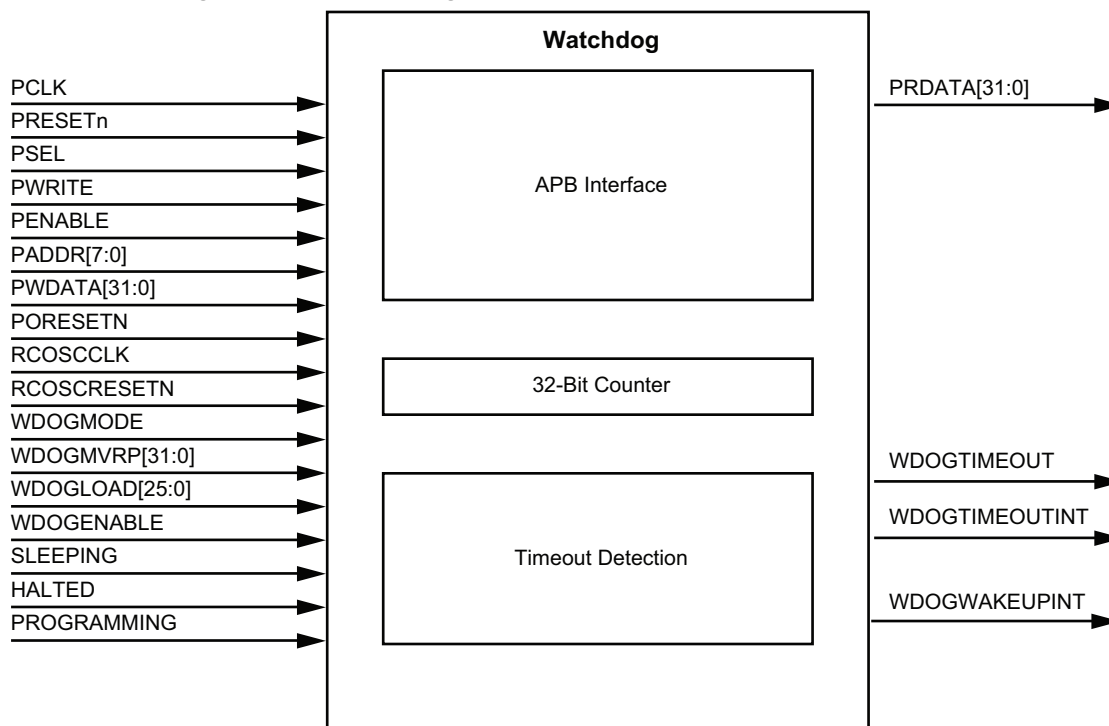
The following sub-sections provide a detailed description of the Watchdog timer.

20.2.1 Architecture Overview

The watchdog timer consists of the following components (as shown in the following figure):

- APB Interface
- 32-Bit Counter
- Timeout Detection

Figure 268 • Block Diagram for the Watchdog Timer



20.2.1.1 APB Interface

The watchdog timer has an APB interface through which the Cortex-M3 processor can access various control and status registers to control and monitor its operation. The APB interface is clocked by the PCLK0 clock signal. Any signals/values that cross between the RCOSCCLK and the PCLK clock domains are synchronized with the watchdog timer.

20.2.1.2 32-Bit Counter

The operation of the watchdog timer is based on a 32-bit down counter that must be refreshed at regular intervals by the Cortex-M3 processor. If not refreshed, the counter will time out. This either causes a system reset or generates an interrupt to the processor, depending on the value of the WDOGMODE bit as defined in the WDOG_CR Register.

The WDOG_CR register is one of the system registers that helps to configure the watchdog timer. In normal operation, the generation of a reset or timeout interrupt by the watchdog timer does not occur because the watchdog timer counter is refreshed on a regular basis.

When the device is powered up, the watchdog timer is enabled with the timeout period set to approximately 10.47 seconds (if Vdd = 1.2 V). The WDOGENABLE bit in the WDOG_CR register controls enabling/disabling of the watchdog timer.

The memory map address for the watchdog timer is 0x40005000-0x40005FFF. The 32-bit counter in the watchdog timer is clocked with the clock signal from the RC Oscillator (RCOSCCLK) which has a frequency of 50 MHz (if Vdd = 1.2 V) with a 5% tolerance.

20.2.1.3 Timeout Detection

A control bit in the WDOGCONTROL register is used to determine whether the watchdog timer generates a reset or an interrupt if a counter timeout occurs. The default setting is reset generation on timeout. When interrupt generation is selected, the WDOGTIMEOUTINT output is asserted on timeout and remains asserted until the interrupt is cleared. When reset generation is selected, the watchdog timer does not directly generate the system reset signal. Instead, when the counter reaches zero, the watchdog timer generates a pulse on the WDOGTIMEOUT output and this is routed to the reset controller to cause it to assert the necessary reset signals. The pulse on the WDOGTIMEOUT output is generated in the RCOSCCLK domain and has duration of one clock cycle.

20.2.2 Port List

The following table lists the ports of the Watchdog Timer module.

Table 633 • Watchdog Timer Interface Signals

Name	Type	Width	Description
WDOGTIMEOUTINT	Output	1	This interrupt is asserted, if the counter reaches zero and interrupt rather than reset generation has been selected on counter timeout.

20.2.3 Details of Operation

This section provides the details of operation of Watchdog timer.

20.2.3.1 Loading and Refreshing the Watchdog Timer

The WDOGLOAD register is used to store the value that is loaded into the counter each time the watchdog timer is refreshed. The six least significant bits of the WDOGLOAD register are always set to 0x3F, irrespective of what value is written to it. This effectively means that there is a lower limit on the value that can be written to the counter. After refreshing, at least 64 RCOSCCLK clock ticks (0.00128ms for Vdd=1.2 V) are required before the counter times out. The purpose of this feature is to prevent a watchdog timer reset/interrupt from occurring immediately after, or during refresh in the case where a very low value has been written to the WDOGLOAD register.

The watchdog timer counter is refreshed by writing the value 0xAC15DE42 to the WDOGREFRESH register. This causes the counter to be loaded with the value in the WDOGLOAD register as defined in the system register block shown in [Table 644](#), page 641.

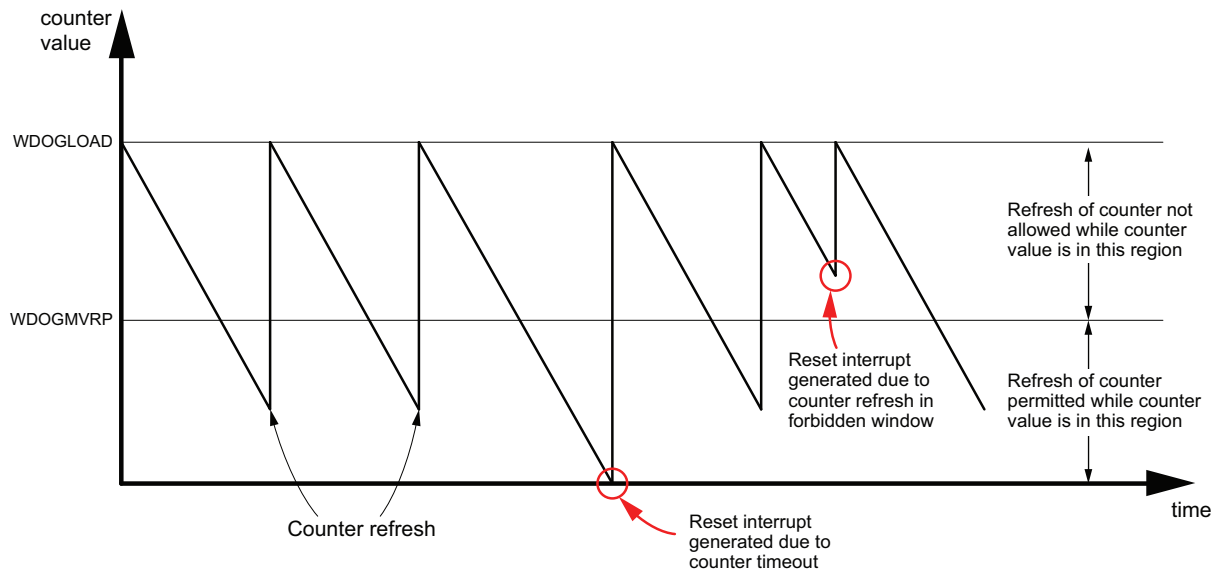
An appropriate value must be written to the WDOGLOAD System register before writing to the WDOGREFRESH register. Forbidden and permitted windows in time regulate when refreshing can occur. The size of these windows is controlled by the value in the WDOGMVRP System register.

When the counter value is greater than the value in the WDOGMVRP, refreshing the watchdog timer is forbidden. If a refresh is executed in these circumstances, the refresh is successful, but a reset or interrupt (depending on Operation mode selected) is also generated. This is shown in the following figure.

When the counter value falls below the level programmed in the WDOGMVRP, refreshing of the watchdog timer is permitted. It is possible to avoid having forbidden and permitted windows by ensuring that the value in the WDOGMVRP is greater than the value in the WDOGLOAD.

The following figure shows how the value of the watchdog timer counter might vary with time.

Figure 269 • Watchdog Timer Counter



Calendar Counter Description

When the TIMEOUT occurs, the watchdog timer counter is reloaded with the WDOGLOAD value. Hence the WDOGLOAD from the system registers block should be programmed such that it should not be lesser than the default value of 0x1800000. This is because if the WDOGLOAD is programmed to a smaller value, each time the WDOGTIMEOUT occurs, the watchdog timer counter is assigned to a lesser value that results in the counter reaching the timeout value within a short period of time and generating the TIMEOUT repeatedly. To avoid such instances, the firmware programs the WDOGLOAD value in the system register to be higher or equal to the default value, which is 0x1800000.

20.2.3.2 Watchdog Timer Behavior During Microcontroller Modes, Device Programming, and Flash*Freeze

This section describes the behavior of the watchdog timer in the Cortex-M3 processor modes and when the device is being programmed.

20.2.3.2.1 Cortex-M3 Processor in Debug State

The halted output from the Cortex-M3 is asserted when the processor is in debug mode and this signal is fed to the Watchdog. When the halted signal is asserted, the watchdog timer counter is halted. This ensures that the watchdog timer timeout-related resets or interrupts do not occur when a system debug session is in progress.

20.2.3.2.2 Cortex-M3 Processor in Sleep Mode

The Cortex-M3 processor can be put into a low-power state by entering sleep mode. The processor exits sleep mode when an interrupt occurs. The watchdog timer can be configured to generate an interrupt if its counter value moves from the permitted to the forbidden window (at the WDOGMVRP level) when the Cortex-M3 processor is in sleep mode. The processor wakes up and refreshes the watchdog timer and then goes back into sleep mode. The WDOGWAKEUPINT output from the watchdog timer is used for this interrupt. The WAKEUPINTEN control bit in the [WDOGCONTROL](#) is used to enable/disable generation of the WDOGWAKEUPINT interrupt, with the default setting being disabled.

20.2.3.2.3 Programming FPGA Fabric/eNVM

The watchdog timer has an input port called PROGRAMMING, which is connected to the watchdog_freeze signal in the microcontroller subsystem (MSS). The watchdog_freeze is asserted by the system controller under certain conditions, such as when programming the eNVM. When the PROGRAMMING port is asserted, the watchdog timer counting is paused. When the PROGRAMMING is de-asserted, the watchdog timer behaves as if it has just come out of reset.

20.2.3.2.4 Flash*Freeze

During Flash*Freeze, the watchdog timer continues to operate unless the Cortex-M3 processor enters Sleep mode. In Sleep mode, the watchdog timer stops counting and holds the current value.

20.2.3.3 Watchdog Timer Interrupts

There are two interrupt outputs in the watchdog timer: WDOGTIMEOUTINT and WDOGWAKUPINT.

20.2.3.3.1 WDOGTIMEOUTINT

This interrupt is asserted when a counter timeout occurs and an interrupt instead of reset generation is selected. This interrupt is connected to the non-maskable interrupt (NMI) input of the Cortex-M3 processor and can also be exposed to the FPGA fabric.

20.2.3.3.2 WDOGWAKEUPINT

This is asserted (if enabled) on crossing the WDOGMVRP level when the SLEEPING input is asserted. This interrupt is mapped to the interrupt request 0 (INTISR [0]) in the Cortex-M3 processor interrupt controller.

20.3 How to Use the Watchdog Timer

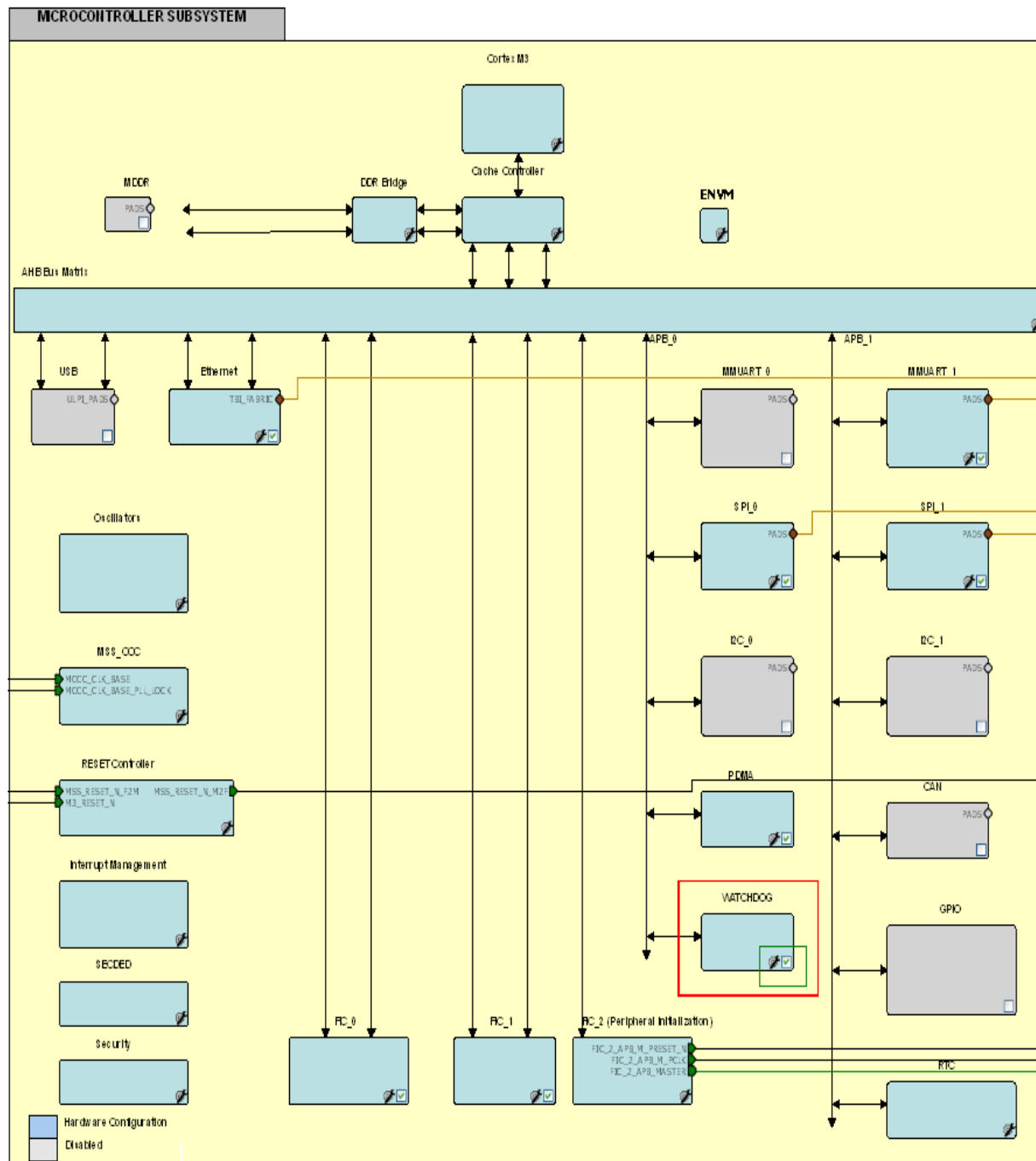
This section describes how to use the watchdog timer in an application.

20.3.1 Design Flow

The following steps are used to enable the WatchDog Timer in the application:

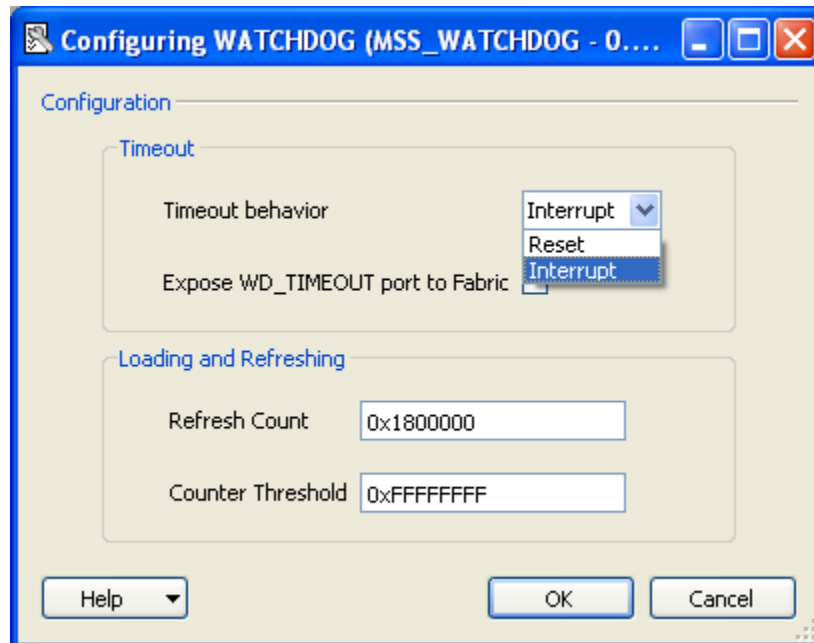
1. Enable the watchdog timer by using the MSS configurator in the application, as shown in the following figure.

Figure 270 • Enabling Watchdog Timer in the Libero SOC Design MSS Configurator



- Clicking **Watchdog Timer** displays the watchdog timer configuration window, as shown in the following figure.

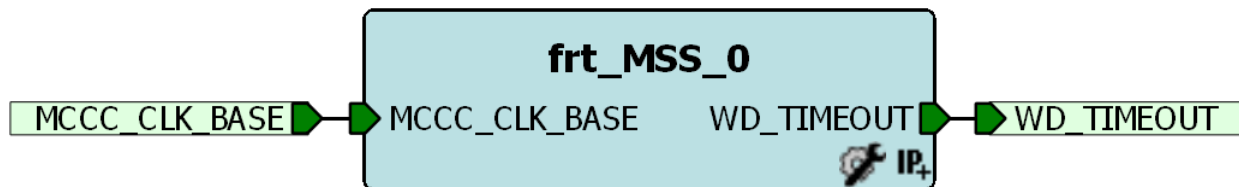
Figure 271 • Watchdog Timer Configuration Window



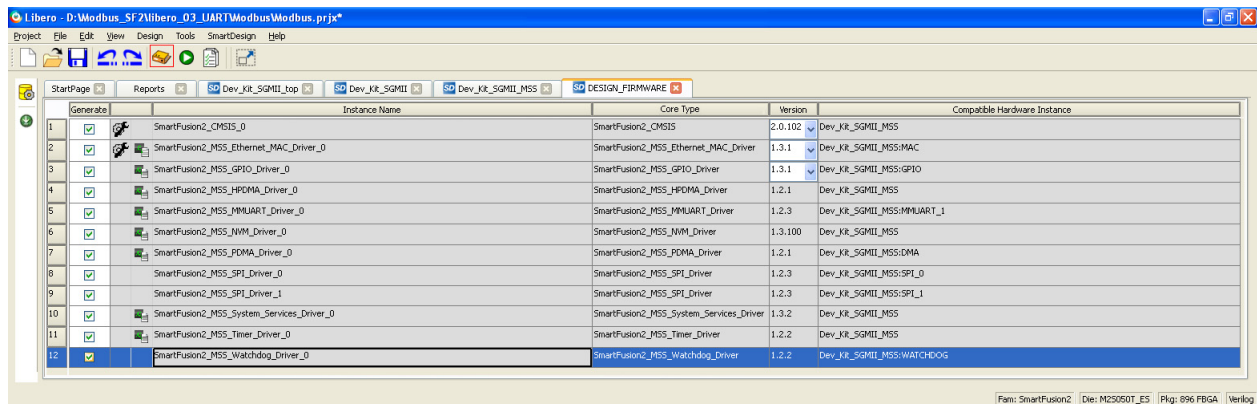
- **Timeout behavior:** The watchdog timer default setting is reset generation on timeout. When interrupt generation is selected, the WDOGTIMEOUTINT output is asserted on timeout and remains asserted until the interrupt is cleared
- **Interrupt Port:** If the Timeout behavior option has been set to interrupt you can expose the WD_TIMEOUT (WDOGTIMEOUTINT) port the FPGA fabric by checking the Expose WD_TIMEOUT (WDOGTIMEOUTINT) port to Fabric check box.
- **Refresh Count:** Use the Refresh Count option to set the WDOGLOAD register value (Flash Bits) at POR or when the device is reset (DEV_RST_N is asserted/de-asserted). Refresh Count value should be higher or equal to the default value, which is 0x1800000.
- **Counter Threshold:** Use the Counter Threshold option to set the value for WDOGMVRP System register. It is possible to avoid having forbidden and permitted windows by ensuring that the value in the WDOGMVRP is greater than the value in the WDOGLOAD. Refer to the [Loading and Refreshing the Watchdog Timer](#), page 631 for a detailed description of forbidden and permitted windows.

- The watchdog timer signals in top level instance are shown in the following figure.

Figure 272 • Watchdog Timer Signals



4. Generate the component by clicking **Generate Component** or by selecting **SmartDesign > Generate Component**. For more information on generation of the component, refer to the [Libero SoC User Guide](#). The firmware driver folder and SoftConsole workspace is included in the project. Click the highlighted **Configure firmware** button as shown in the following figure to find the watchdog timer drivers.

Figure 273 • RTC Driver User Guide


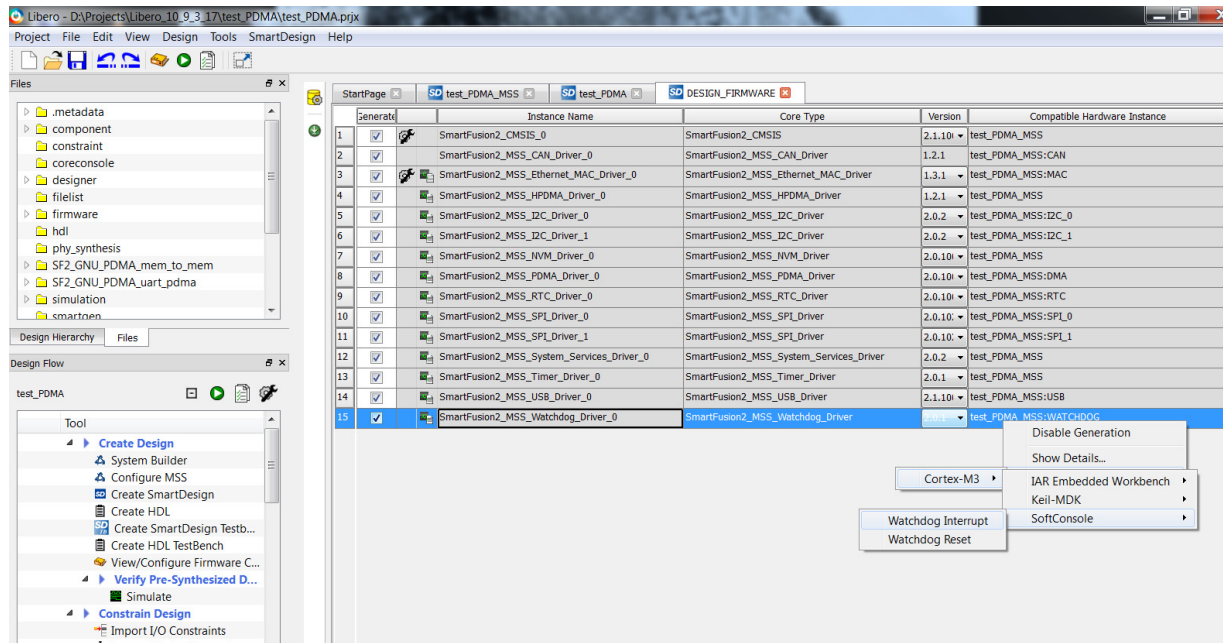
5. Click **Generate Bitstream** under **Program Design** to complete *.fdb file generation.
6. Double-click **Export Firmware** under **Handoff Design for Firmware Development** in the Libero SoC design flow window to generate the SoftConsole **Firmware Project**. The SoftConsole folder contains the mss_watchdog firmware driver. The firmware driver, mss_watchdog (mss_watchdog.h), which provides a set of functions for controlling the watchdog timer can also be downloaded from the Microsemi firmware catalog. The following table lists the APIs for Watchdog Timer. For more information on the APIs, refer to the **SmartFusion2_MSS_Watchdog_Driver_UG**.

Table 634 • Watchdog Timer APIs

Category	API	Description and Usage
Initialization	MSS_WD_init()	Initializes Watchdog Timer
Reading the watchdog timer current value and status	MSS_WD_current_value()	Returns the current value of the watchdog's down counter
	MSS_WD_status()	Returns the status of the watchdog
Refreshing the watchdog timer value	MSS_WD_reload()	Causes the watchdog to reload its down counter timer with the load value
	MSS_WD_timeout_occured()	Reports the occurrence of a timeout event
	MSS_WD_clear_timeout_event()	Clears the hardware's report of a time out event
Time-out and wake-up interrupts control	MSS_WD_enable_timeout_irq()	Enables the watchdog's time out interrupt
	MSS_WD_disable_timeout_irq()	Disables the generation of the NMI interrupt
	MSS_WD_enable_wakeup_irq()	Enables the SmartFusion2 wakeup interrupt
	MSS_WD_clear_wakeup_irq()	Clears the wakeup interrupt
	MSS_WD_disable_wakeup_irq()	Disables the SmartFusion2 wakeup interrupt
	MSS_WD_clear_timeout_irq()	Clears the watchdog's time out interrupt

- For more information on Watchdog usage, the sample projects are available and can be generated, as shown in the following figure.

Figure 274 • Watchdog Timer Examples



20.3.2 Watchdog Timer Use Models

20.3.2.1 Use Model 1

The steps below are used to generate a reset by the watchdog timer on timeout.

- Enable the watchdog timer in MSS configurator of your Libero project.
- Select timeout behavior as reset in watchdog timer configuration window.
- Initialize the watchdog timer in the SoftConsole application using `MSS_WD_init()` function.
- Use `MSS_WD_reload()` function to reload its down counter timer with the load value configured through the call to `WD_init()`. This function must be called regularly to avoid a system reset.
- Use `MSS_WD_timeout_occured()` to report the occurrence of a timeout event. It can be used to detect if the reset is resetting as part of a watchdog timeout.
- Call `MSS_WD_clear_timeout_event()` function after a call to `MSS_WD_timeout_occured()` function to clear the hardware's report of a time out event.

20.3.2.2 Use Model 2

The steps below are used to generate a watchdog timer interrupt on timeout.

- Enable the watchdog timer in MSS configurator of your Libero project.
- Select timeout behavior as interrupt in the watchdog timer configuration window.
- Initialize the watchdog timer in SoftConsole application using `MSS_WD_init()` function.
- Enable the watchdog timer timeout interrupt using `MSS_WD_enable_timeout_irq()` function.
- Use `MSS_WD_reload()` function to reload its down counter timer with the load value configured through the call to `WD_init()`.
- Use `MSS_WD_clear_timeout_irq()` function in interrupt handler to clear the watchdog timeout interrupt.

Notes:

- The watchdog timer cannot be disabled through software. Only the interrupts can be disabled.
- The MSS Watchdog timer does not support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for more information.

20.4 Watchdog Timer Register Map

The following table summarizes the watchdog timer register interface. Detailed description of the registers is given in [Watchdog Timer Configuration Register Bit Definitions](#), page 638. The base address for the register details resides at 0x40005000 and extends to the address 0x40005FFF in the Cortex-M3 processor memory map.

Table 635 • Watchdog Timer Register Interface Summary

Register Name	Address Offset	R/W	Reset Source	Description
WDOGVALUE	0x00	R	WDOGLOAD	Current value of the counter This register is reset with the value in the WDOGLOAD system register.
WDOGLOAD	0x04	R	WDOGLOAD	Load value for the counter This register is reset with the value in the WDOGLOAD system register.
WDOGMVRP	0x08	R	WDOGMVRP	Maximum value for which refreshing is permitted. This register is reset with the value in the WDOGMVRP system register.
WDOGREFRESH	0x0C	W	N/A	Writing the value 0xAC15DE42 to this register causes the counter to be updated with the value in the WDOGLOAD register.
WDOGENABLE	0x10	R	WDOGENABLE	Watchdog timer enables register This register is reset with the value in the WDOGENABLE bit in the WDOG_CR system register.
WDOGCONTROL	0x14	R/W	[31:3] and [1:0]=0X0 Bit 2=WDOGMODE	Control register Bit 2 of this register is reset with the value of the WDOGMODE bit in the WDOG_CR system register.
WDOGSTATUS	0x18	R	0X0	Status register
WDOGRIS	0x1C	R/W	0X0	Raw interrupt status

20.4.1 Watchdog Timer Configuration Register Bit Definitions

The watchdog timer registers are described in detail in the following tables.

Table 636 • WDOGVALUE

Bit Number	Name	Reset Value	Description
[31:0]	WDOGVALUE	WDOGLOAD	This register contains the current value of the counter in the watchdog timer. This register is reset with the value in the WDOGLOAD system register.

Table 637 • WDOGLOAD

Bit Number	Name	Reset Value	Description
[31:0]	WDOGLOAD[31:6]	WDOGLOAD	This register contains the upper 26 bits load value for the counter in the watchdog timer. This will be updated by the WDOGLOAD[25:0] bits of the system registers. The least significant bits of the register always have the value of 0x3F.

Table 638 • WDOGMVRP

Bit Number	Name	Reset Value	Description
[31:0]	WDOGMVRP	WDOGMVRP	<p>This register contains the maximum value for which refreshing is permitted.</p> <p>If the watchdog timer is refreshed (by writing to the WDOGREFRESH register) while the counter value is greater than the value in the WDOGMVRP, then a reset/interrupt is generated. This read only register is loaded with the user flash bit value written in the WDOGMVRP[31:0] system register.</p>

Table 639 • WDOGREFRESH

Bit Number	Name	Reset Value	Description
[31:0]	WDOGREFRESH	N/A	<p>This is a write only register which reads as zero. Writing the value 0xAC15DE42 to this register causes the counter to be refreshed with the value in the WDOGLOAD register.</p> <p>If this register is written to, while the current value of the counter is greater than the value in the WDOGMVRP register, the counter is refreshed and a reset or timeout interrupt is generated (depending on the MODE bit of the WDOGCONTROL). While the counter value is greater than the WDOGMVRP, there is effectively a time window in which it is forbidden to refresh the watchdog timer.</p> <p>When the counter is in between the WDOGMVRP level and 0, the watchdog timer is in a time window where it is permitted for it to be refreshed.</p> <p>It is possible to avoid having the forbidden and permitted time windows for refreshing the watchdog timer by setting the value of the WDOGMVRP to a value greater than that stored in the WDOGLOAD.</p>

Table 640 • WDOGENABLE

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0x0	Reserved
0	ENABLE	WDOGENABLE	<p>Enable bit for watchdog timer.</p> <p>This bit holds the value of the USER FLASH bit written in the WDOGENABLE bit of the WDOG_CR system register.</p>

Table 641 • WDOGCONTROL

Bit Number	Name	Reset Value	Description
[31:3]	Reserved	0	To provide compatibility to the future products, the value of a reserved bit should be preserved across a read-modify-write operation.
2	MODE	WDOGMODE	Operation mode for the watchdog timer. 0: reset is generated if counter reaches zero. 1: interrupt is generated if counter reaches zero. This read only register holds the value of user flash bit written in the WDOGMODE bit of the WDOG_CR system register.
1	WAKEUPINTEN	0	0: The WDOGWAKEUPINT interrupt generation is disabled. 1: The WDOGWAKEUPINT interrupt generation is enabled.
0	TIMEOUTINTEN	0	0: The WDOGTIMEOUTINT interrupt generation is disabled. 1: The WDOGTIMEOUTINT interrupt generation is enabled.

Table 642 • WDOGSTATUS

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	To provide the compatibility to the future products, the value of a reserved bit should be preserved across a read-modify-write operation
0	REFRESHSTATUS	0	Refresh status 0: The counter is in forbidden window, refresh should not be initiated. 1: The counter in is permitted window, refresh is allowed.

Table 643 • WDOGRIS

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
1	WAKEUPRS	0	Raw status of wakeup interrupt. Writing '1' to this bit clears the bit. Writing '0' has no effect.
0	TIMEOUTRS	0	Raw status of counter timeout interrupts. Writing '1' to this bit clears the bit. Writing '0' has no effect.

20.5 SYSREG Control Registers

In addition to the specific watchdog timer registers mentioned in [Table 635](#), page 638, the registers mentioned in the following table also control the behavior of the watchdog timer. These registers are located in the System Register Block and are listed here for clarity. Refer to the SYSREG section of [System Register Block](#), page 670 for a detailed description of each register and bit.

Table 644 • Watchdog Timer SYSREG

Register Name	Register Type	Flash Write Protect	Reset Source	Description
WDOG_CR	RW-P	Register	Poreset_n	Bit 0 of this register is WDOGENABLE. This goes as the Enable bit for the watchdog timer module. The status of this bit can be monitored in the WDOGENABLE register. Bit 1 of this register is WDOGMODE. This bit is Reset/Interrupt mode selection bit from the system register. This value can be read from the WDOGCONTROL register within the watchdog timer module.
WDOGLOAD	RO-P		Poreset_n	Bits [25:0] of this register contain the upper 26-bits of the WDOGLOAD value register.
WDOGMVRP	RO-P		Poreset_n	This register contains the WDOGMVRP value.

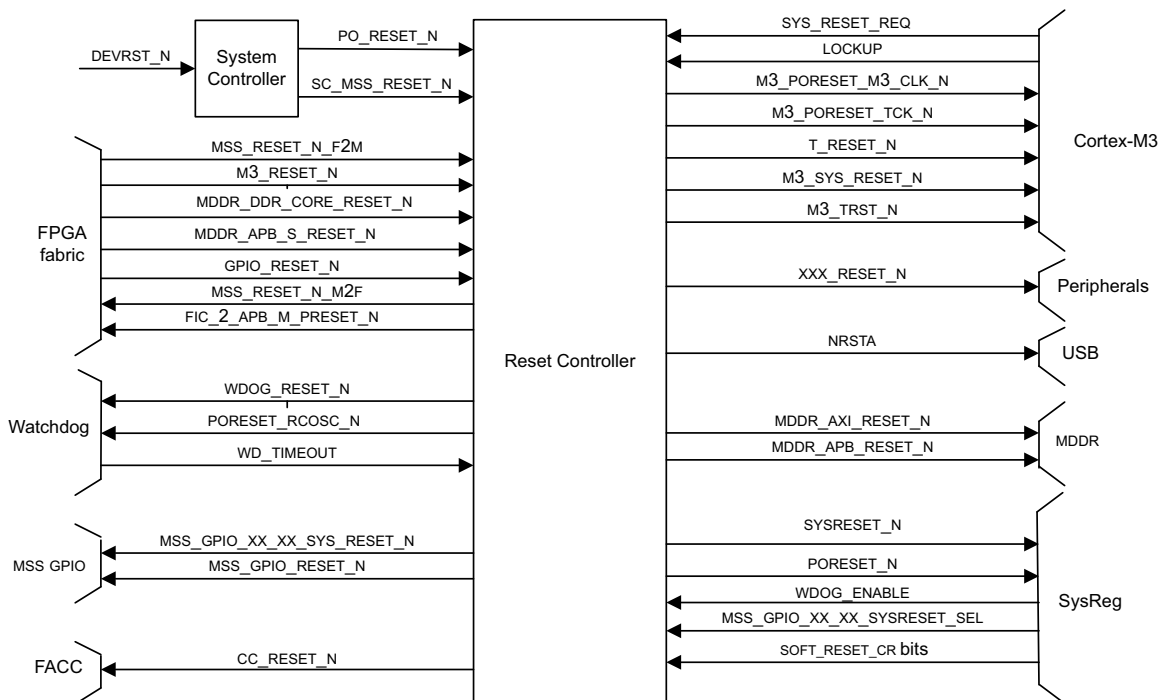
Note: Register Types RW-P, RO-P are explained in [System Register Block](#), page 670.

21 Reset Controller

The Reset Controller manages the asynchronous reset requests coming from various sources and generates a synchronous reset for the entire MSS or individual resets to the MSS sub-blocks and user logic in the FPGA fabric.

The Reset Controller drives resets to various modules of the SmartFusion2 devices, such as the Cortex-M3 processor, MDDR subsystem, Watchdog Timer, FPGA fabric, MSS GPIO, clock controller, SYSREG, and peripherals. The following figure shows the Reset Controller with various reset inputs/outputs from/to various MSS blocks.

Figure 275 • Reset Signals Distribution in SmartFusion2 Devices



Microsemi recommends to use the CoreResetP IP for initializing the user design in SmartFusion2 Devices. The CoreResetP handles sequencing of reset signals in SmartFusion2 devices. It is available in the Libero System-on-Chip (SoC) IP catalog. The System Builder is a powerful design tool within the Libero SoC Design Environment that helps you capture your system-level requirements and produces a design implementing those requirements. A very important function of the System Builder is the automatic creation of the 'initialization' sub-system (all required cores are instantiated, and connections are made automatically).

21.1 Functional Description

21.1.1 Power-On Reset Generation Sequence

The following figure shows the conceptual block diagram of power-on reset generation. The POR generator block in System Controller generates a power-on reset signal, PO_RESET_N.

Figure 276 • Conceptual Block Diagram of Power-On Reset Generation

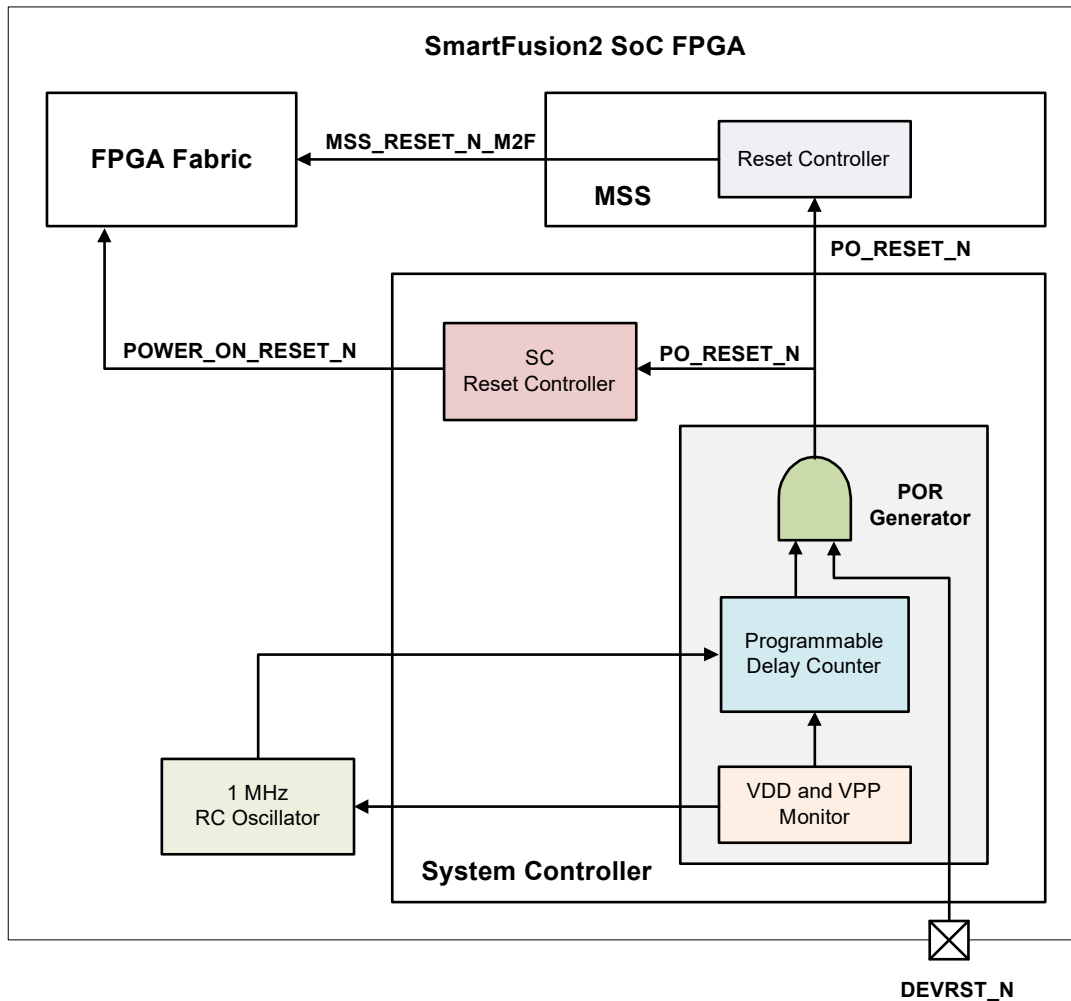
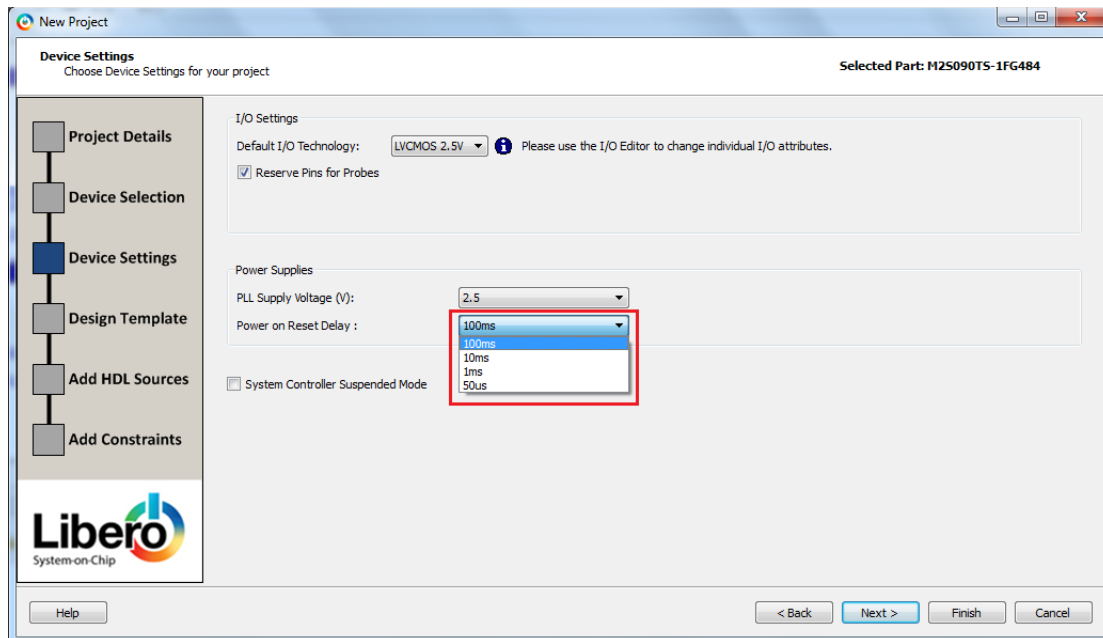


Figure 279, page 645 shows the power up to functional time sequence diagram. On power-up, the VDD and VPP monitor blocks in the POR generator block assert a power-on reset signal, PO_RESET_N. If the VDD and VPP supplies reach their threshold point (VDD ~ 0.9 V, VPP ~ 0.9 V), the 1 MHz RC Oscillator is turned-on, which provides the clock to the programmable delay counter. The delay can be configured to 50 μ s, 1 ms, 10 ms, or 100 ms in the New Project window (Device Settings) while creating the Libero SoC project as shown in the following figure. You can also access and change this setting after the project has been created from the Project Settings window (Project > Project Settings...). The delay setting (Power on Reset Delay) get implemented in the design while generating bitstream.

Figure 277 • Power on Reset Delay Configuration

The delay counter is used to generate the power supply rise time. All power supplies must be stable within the configured Power on Reset Delay. When the counter reaches its maximum value, the PO_RESET_N signal is de-asserted. Upon de-assertion of the PO_RESET_N signal, the 1 MHz RC oscillator is gated off and the 50 MHz RC oscillator is enabled and the System Controller starts operating at 50 MHz clock. Then System Controller starts the initialization sequence of I/O banks, MSS, and FPGA Fabric Subsystem.

The POWER_ON_RESET_N signal is generated from the PO_RESET_N signal and can be used in the user design as a reset for the FPGA fabric logic. It is an active low output signal. It is made available by instantiating the SYSRESET macro from the Libero SoC IP catalog in SmartDesign or by instantiating the SYSRESET macro directly in the HDL file. The following figure shows a block symbol of the SYSRESET macro that exposes the POWER_ON_RESET_N signal.

Figure 278 • SYSRESET Macro

POWER_ON_RESET_N asserts on the following events:

- Power-up event
- Assertion of DEVRST_N
- Completion of programming
- Completion of zeroization

A dedicated input-only reset pad (DEVRST_N) is present on all the SmartFusion2 devices, which cause assertion to the PO_RESET_N signal. If an external reset circuit is connected to the DEVRST_N pin, it increases the power up to functional time due to the delays that the external reset device does add.

DEVRST_N is an asynchronous reset pin and must be asserted only when the device is unresponsive due to some unforeseen circumstances. It is not recommended to assert the DEVRST_N pin during programming operation, which might cause severe consequences including corrupting the device

configuration. For more details on DEVRST_N timing information, refer to the [DS0451: IGL002 and SmartFusion2 Datasheet](#).

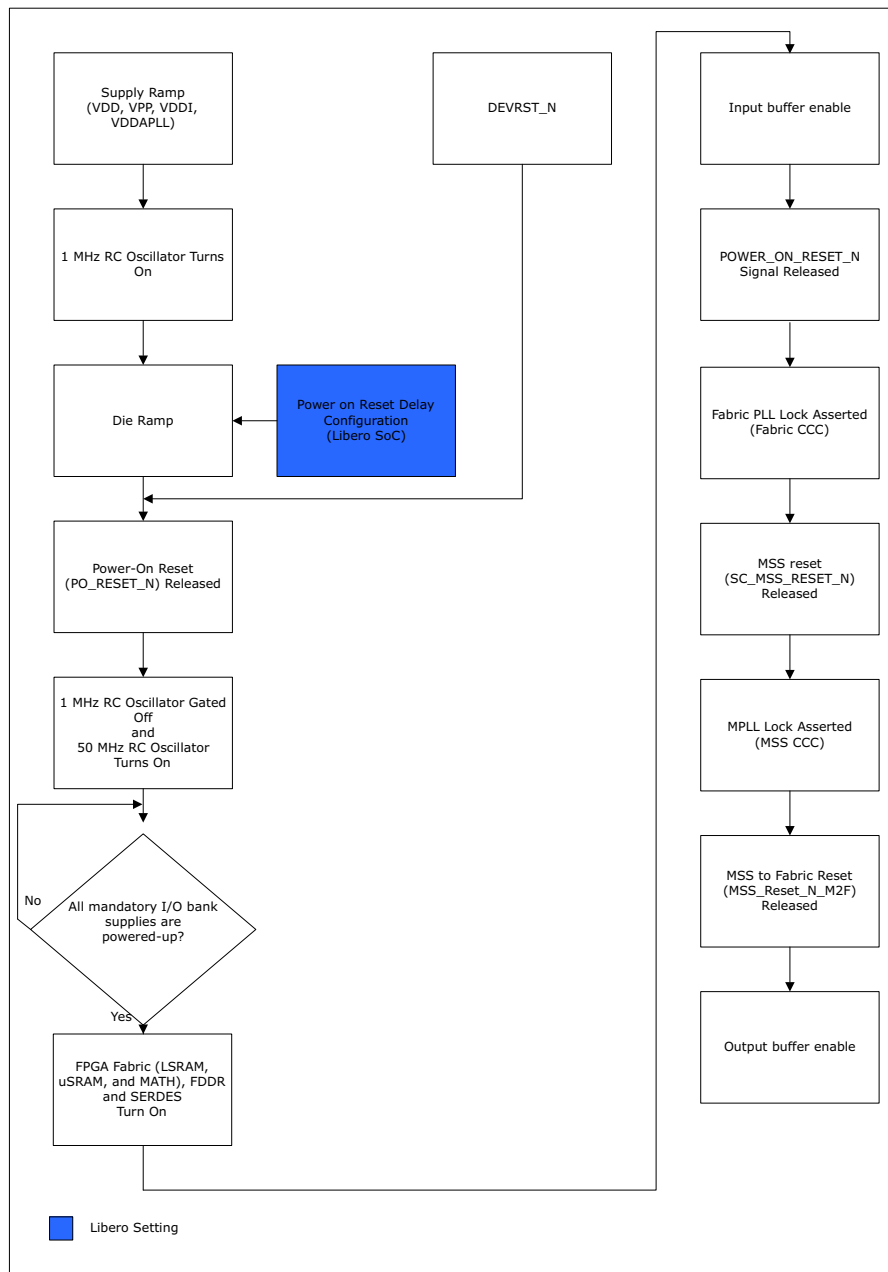
Asserting DEVRST_N does not enable the delay counter (Power on Reset Delay) in the POR circuitry.

The delay counter is operational only at power-up. When DEVRST_N is low, all user I/Os are fully tri-stated. Although, the JTAG I/Os are still enabled, they cannot be used as the TAP controller is in reset. The SYSRESET macro is not required to be instantiated to enable the DEVRST_N pin in the user design. DEVRST_N is a dedicated input-only reset pad available on all the SmartFusion2 devices.

21.1.2 Power-Up to Functional Time Sequence

The following figure shows the power up to functional time sequence diagram.

Figure 279 • Power up to Functional Time Sequence Diagram



The power up to functional sequence is as follows:

- Supply Ramp (VDD, VPP, VDDI, and VDDAPLL) - There is no specific power up or power down sequencing requirement for SmartFusion2 devices. The I/O banks can be brought-up in any order, before or after the core voltage. However, the device is only functional if all I/O bank supplies are powered-up.
- On power-up, the POR generator block asserts the PO_RESET_N signal, which is not accessible for users.
- The 1 MHz RC Oscillator is turned-on, which provides the clock to the programmable delay counter. When the counter reaches its maximum value, the PO_RESET_N signal is de-asserted.
- The 1 MHz RC oscillator is gated off and the 50 MHz RC oscillator is enabled and the System Controller starts operating at 50 MHz clock.
- All mandatory I/O bank supplies must be powered-up. For all the IGLOO®2 devices, some of the bank supplies (VDDIx) must always be powered, even if associated bank I/Os are in unused condition. For the list of mandatory I/O bank supplies, refer to Table 2 and Table 3 in the [AC393: SmartFusion2 and IGLOO2 Board Design Guidelines Application Note](#).
- FPGA fabric (LSRAM, uSRAM, and MATH), FDDR, and SERDES are turned-on.
- Input buffer is enabled.
- POWER_ON_RESET_N signal (generated from the PO_RESET_N signal) is released. This signal can be used in the design as a reset for the FPGA fabric logic.
- Fabric PLL (Fabric CCC) Lock is asserted.
- MSS reset (SC_MSS_RESET_N) is released.
- MPLL (MSS CCC) Lock is asserted.
- MSS to Fabric Reset (MSS_RESET_N_M2F) is released.
- Output buffer is enabled.

The Reset Controller Configurator in the MSS Configurator enables the user to expose the MSS_RESET_N_M2F signal to the fabric. Refer to [How to Use the Reset Controller](#), page 666 for more information. In order to simplify the task of initializing a user design in SmartFusion2 devices, Microsemi provides a CoreResetP soft Reset Controller IP. The CoreResetP handles sequencing of reset signals in SmartFusion2 devices. The CoreResetP generates a fabric reset signal whenever POWER_ON_RESET_N is asserted or MSS_RESET_N_M2F is asserted. It is available in the Libero SoC IP catalog. Refer to [CoreResetP Soft Reset Controller](#), page 660 for more information.

Note: Microsemi recommends using the System Builder which automatically creates the 'initialization' sub-system (all required cores are Instantiated, and connections are made automatically).

21.2 Power-Up to Functional Time Data

This section describes power-up to functional time sequence and provides timing numbers based on DEVRST_N assertion and VDD ramp up.

21.2.1 Parameters Used for Obtaining Power-Up to Functional Time Data

This section describes the parameters used for obtaining power-up to functional time data. Following are the test conditions:

- Power on reset delay setting: 1 ms
- Supply ramp rate: 5 μ s
- Measurement temperature: 25°C

Power-on reset delay setting indicates how long VDD takes to ramp up. The programmable delay counter starts counting based on the power-on reset delay setting, oscillator frequency, and period variability to ensure that the supplies have reached their minimum operating levels.

Supply ramp rate indicates the ramp up rate of on-board VDD core voltage, VPP charge pump voltage, and VDDA phase-locked loop (PLL) analog voltage supplies.

Note: In all the test cases, I/Os are configured as LVCMOS25, which is the default I/O standard in Libero along with the other default I/O attribute settings.

21.2.2 VDD Power-Up to Functional Time

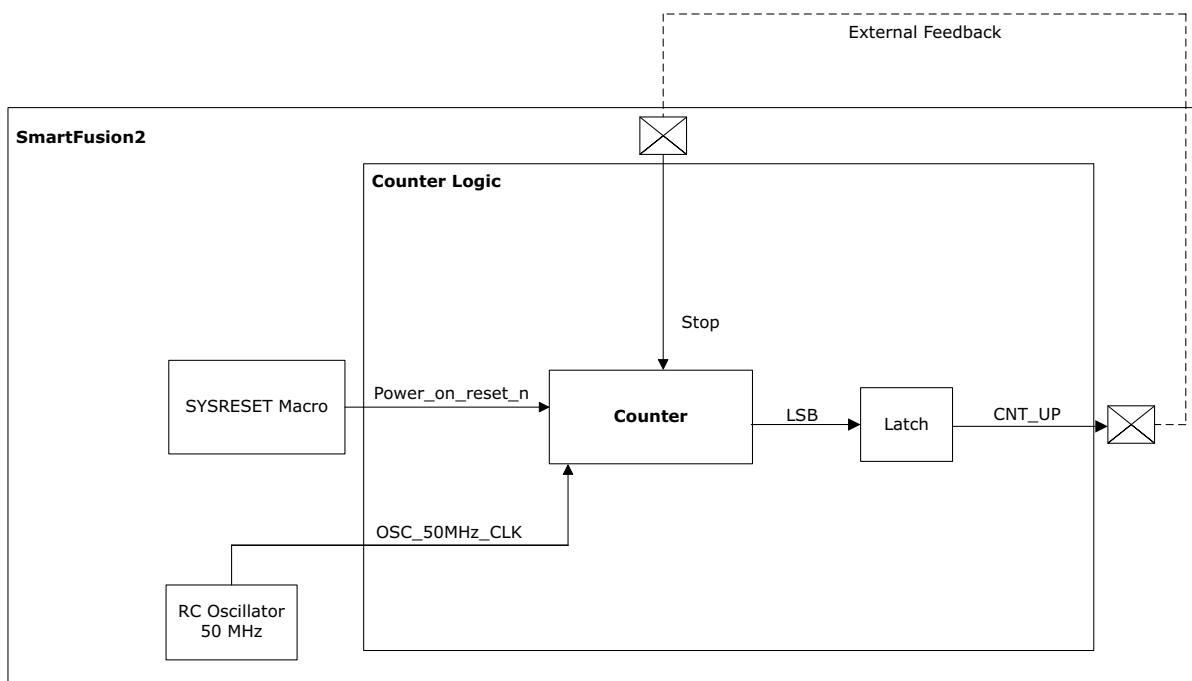
The core supply voltage VDD is connected to the appropriate source and VDD is monitored by the power-on reset circuitry to check if it reaches the minimum threshold value and initiates the system controller to release the device from reset. This scenario provides power-up to functional time data when only the FPGA fabric and the FPGA I/O are used with all supplies ramped up except VDD, which is ramped up at the last. The required power-on reset delay is set using the Libero SoC tool.

The design uses a fabric counter that starts to operate when POWER_ON_RESET_N is deasserted. The LSB of the counter output is connected to a latch and given to an output buffer, which is then connected to an input buffer of the fabric using external loopback. This input is used for stopping the counter from incrementing. The counter stops as soon as the counter's LSB bit transitions to logic HIGH. The power-up to functional time is measured from the VDD supply ramp to transition of the fabric buffer output.

The following figure shows the characterization test design setup used for obtaining the VDD power-up to functional timing values.

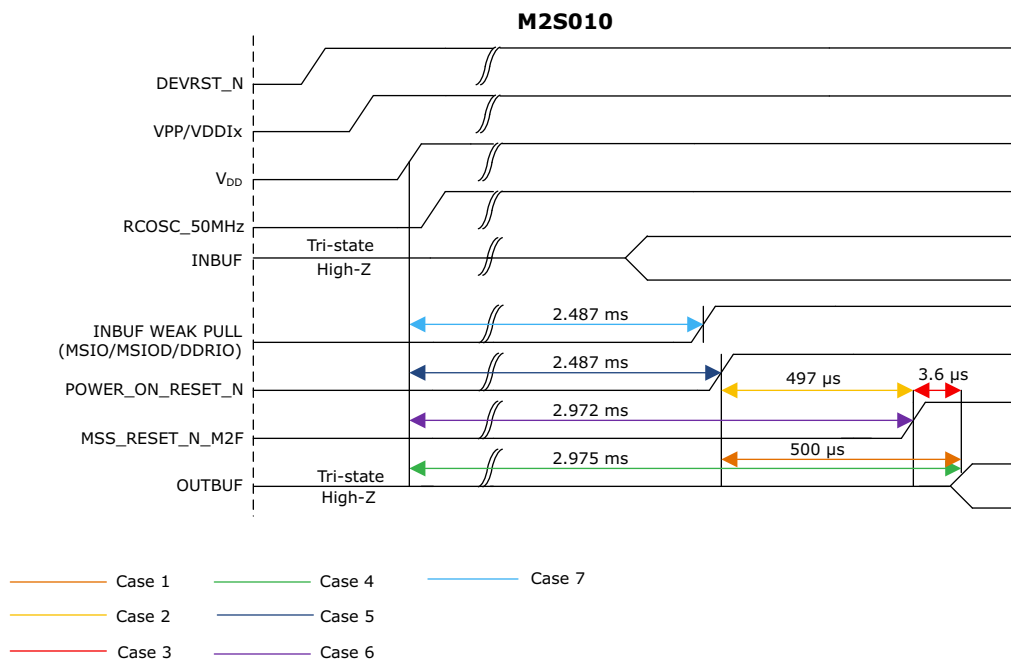
Note: In this test design setup, PLL is not used and instead clock for the fabric is directly fed from the RC oscillator. If PLL is used in the design then the Power-up to functional time will get impacted because of PLL lock assertion time.

Figure 280 • VDD Power-up to Functional Time Design Setup



The following figure shows the behavior of different signals when VDD is ramped with a power on reset delay of 1 ms from 0 V to its minimum threshold level and MSS is used with VDD = 1.2 V, VDDI = 2.5 V, Tj = 25 °C, and power on reset delay setting = 1 ms.

Figure 281 • VDD Power-up to functional timing diagram



The following table lists power-up to functional time data of M2S005, M2S010, M2S025, M2S050, M2S060, M2S090, and M2S150 devices with MSS clock ranging from 3 MHz to 166 MHz.

Note: The timing numbers shown in the below table is for worst case condition.

Table 645 • VDD Power-Up to Functional Time

Test Case	Start Point	End Point	Description	Power-Up to Functional Time (μs)						
				005	010	025	050	060	090	150
Case1	POWER_ON_RESET_N	Output available at I/O	Fabric to output	647	500	531	483	474	524	647
Case 2	POWER_ON_RESET_N	MSS_RESET_N_M2F	Fabric to MSS	644	497	528	480	468	518	641
Case 3	MSS_RESET_N_M2F	Output available at I/O	MSS to output	3.6	3.6	3.6	3.4	4.9	4.8	4.8
Case 4	VDD	Output available at I/O	VDD at its minimum threshold level to output	3096	2975	3012	2959	2869	2992	3225
Case 5	VDD	POWER_ON_RESET_N	VDD at its minimum threshold level to fabric	2476	2487	2496	2486	2406	2563	2602
Case 6	VDD	MSS_RESET_N_M2F	VDD at its minimum threshold level to MSS	3093	2972	3008	2956	2864	2987	3220

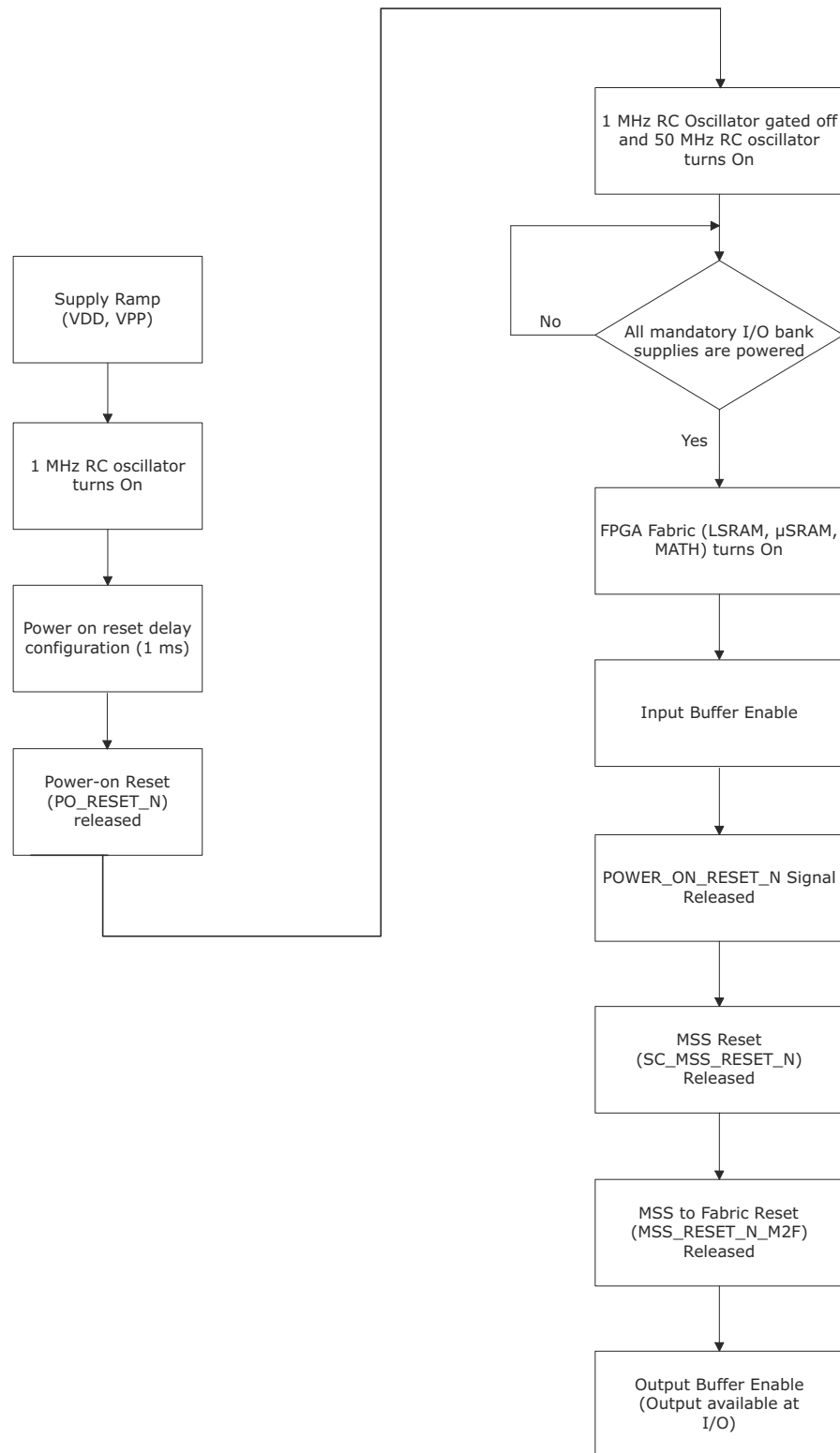
Table 645 • VDD Power-Up to Functional Time (continued)

Test Case	Start Point	End Point	Description	Power-Up to Functional Time (μs)						
				005	010	025	050	060	090	150
Case 7	V _{DD}	DDRIO input buffer weak pull	V _{DD} at its minimum threshold level to input buffer weak pull	2500	2487	2509	2475	2507	2519	2617
	V _{DD}	MSIO input buffer weak pull	V _{DD} at its minimum threshold level to input buffer weak pull	2504	2491	2510	2478	2517	2525	2620
	V _{DD}	MSIOD input buffer weak pull	V _{DD} at its minimum threshold level to input buffer weak pull	2479	2468	2493	2458	2486	2499	2595

Note: Time taken for different power-on reset delay settings can be calculated using the following equation:
 (Test case – 2000 μs) + 2 × Power on Reset Delay setting.

The following figure shows stages that contribute to VDD power-up to functional time for SmartFusion2.

Figure 282 • VDD Power-Up to Functional Time Flow



Note: Power-up to functional time depends on power-on reset delay setting, 1 MHz oscillator frequency, and period variability. At times, it is approximately equal to twice the power-on reset delay setting.

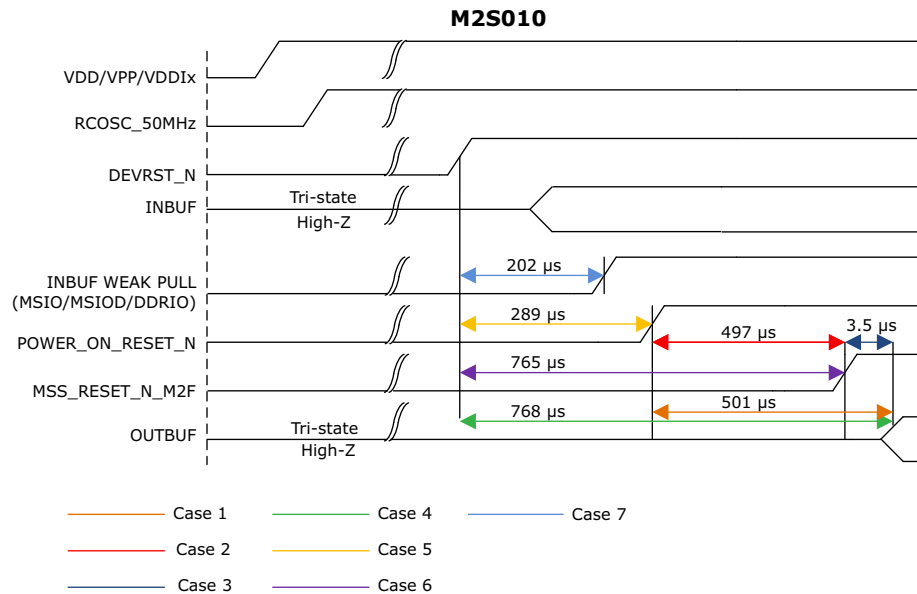
21.2.3 DEVRST_N Power-Up to Functional Time

This scenario provides power-up to functional time data with respect to DEVRST_N when the FPGA fabric, the FPGA I/O, and external oscillator are used. The design setup is same as the VDD power-up to functional time, as shown in the following figure.

Note: It is not recommended to assert DEVRST_N pin during programming (including eNVM), as it corrupts the device configuration. For more information on proper usage of the DEVRST_N pin, see the [AC393: Board Design Guidelines for SmartFusion2 SoC and IGLOO2 FPGAs Application Note](#).

The following figure shows the behavior of different signals when DEVRST_N is asserted and MSS is used with VDD = 1.2 V, VDDI = 2.5 V, and T_j = 25 °C.

Figure 283 • DEVRST_N Power-Up to Functional Timing



The following table lists power-up to functional time of M2S005, M2S010, M2S025, M2S050, M2S060, M2S090, and M2S150 devices with MSS clock ranging from 3 MHz to 166 MHz.

Note: The timing numbers shown in the below table is for worst case condition.

Table 646 • DEVRST_N Power-Up to Functional Time

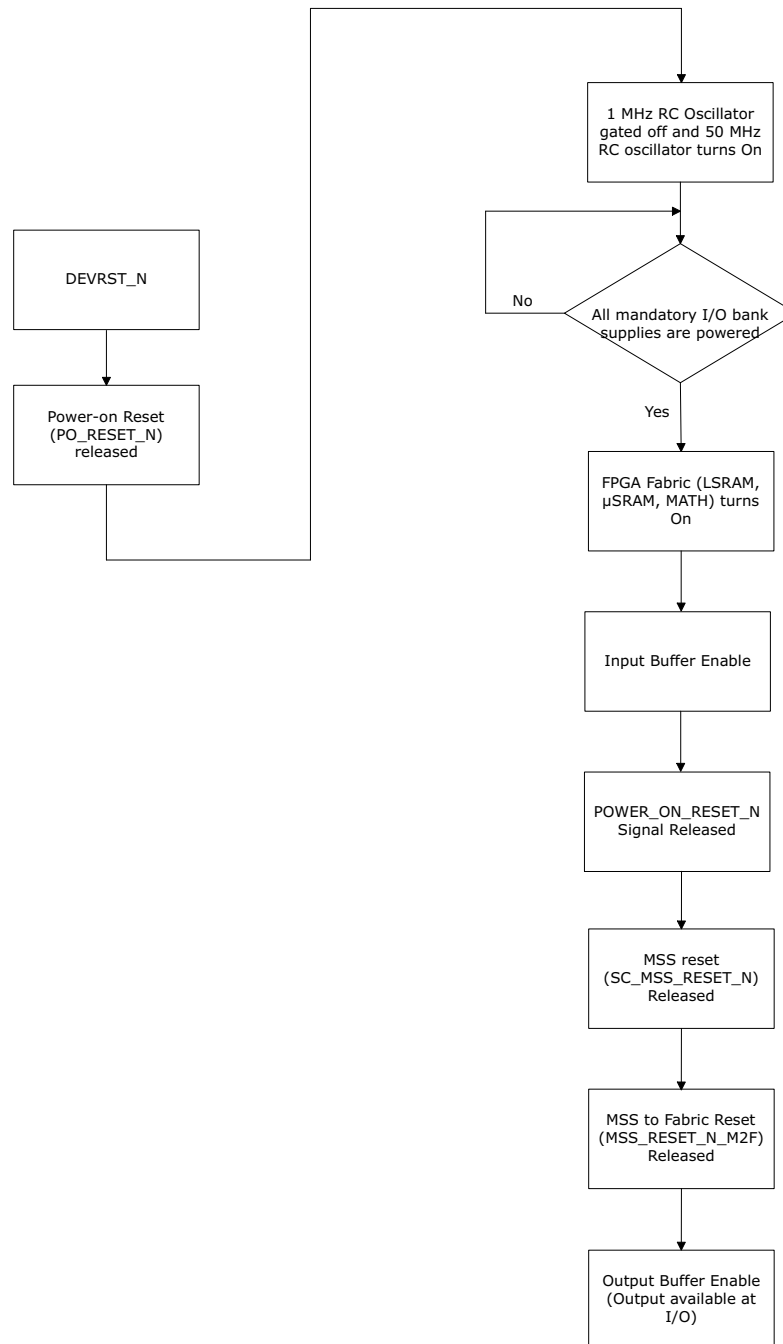
Test Case	Start Point	End Point	Description	Power-Up to Functional Time (μs)						
				005	010	025	050	060	090	150
Case1	POWER_ON_RESET_N	Output available at I/O	Fabric to output	518	501	527	521	422	419	694
Case 2	POWER_ON_RESET_N	MSS_RESET_N_M2F	Fabric to MSS	515	497	524	518	417	414	689
Case 3	MSS_RESET_N_M2F	Output available at I/O	MSS to output	3.5	3.5	3.5	3.3	4.8	4.8	4.8
Case 4	DEVRST_N	Output available at I/O	DEVRST_N to output	706	768	715	691	641	635	871
Case 5	DEVRST_N	POWER_ON_RESET_N	DEVRST_N to fabric	234	289	216	213	237	234	219
Case 6	DEVRST_N	MSS_RESET_N_M2F	DEVRST_N to MSS	702	765	712	688	636	630	866

Table 646 • DEVRST_N Power-Up to Functional Time (continued)

Test Case	Start Point	End Point	Description	Power-Up to Functional Time (μs)						
				005	010	025	050	060	090	150
Case 7	DEVRST_N	DDRIO input buffer weak pull	DEVRST_N to input buffer weak	208	202	197	193	216	215	215
	DEVRST_N	MSIO input buffer weak pull	DEVRST_N to input buffer weak	208	202	197	193	216	215	215
	DEVRST_N	MSIOD input buffer weak pull	DEVRST_N to input buffer weak	208	202	197	193	216	215	215

The following figure shows the stages that contribute to DEVRST_N power-up to functional time for SmartFusion2.

Figure 284 • DEVRST_N Power-up to Functional Time Flow



Note: All timing numbers in [Table 645](#), page 648 and [Table 646](#), page 651 are for worst case conditions.

21.2.4 Power-On Reset

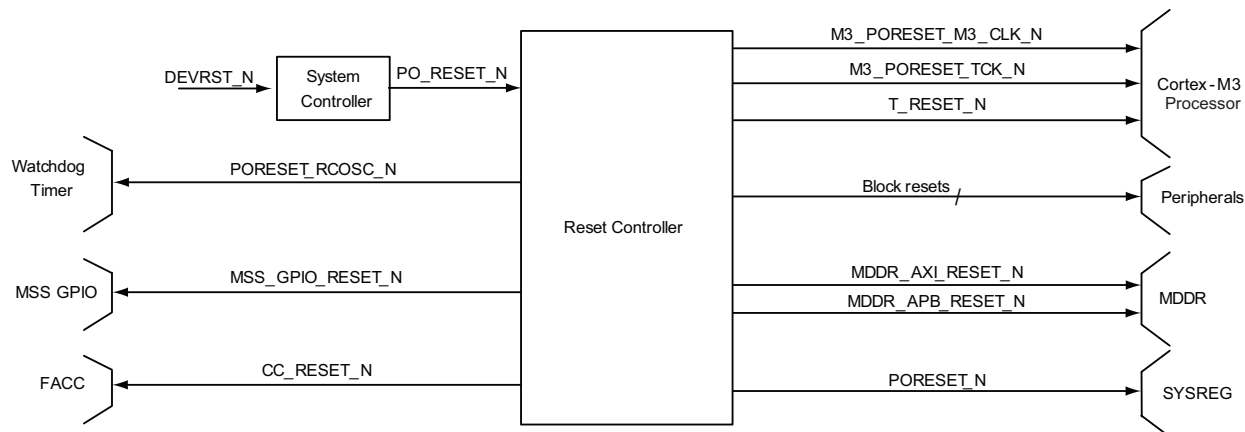
The Reset Controller receives a power-on reset signal, PO_RESET_N, from the System Controller, which is a cold reset signal. Its assertion initializes the SmartFusion2 device to its default reset state.

PO_RESET_N signal is fed to the system register block (SYSREG). The PO_RESET_DETECT bit of the RESET_SOURCE_CR register (defined in [Table 648](#), page 669) in the SYSREG block is set or reset depending on the PORESET_N signal.

The PORESET_N signal is a synchronized version of the PO_RESET_N signal on M3_CLK.

The Reset Controller generates different synchronized resets to the MSS and the FPGA fabric on the assertion of PO_RESET_N, as shown in the following figure.

Figure 285 • Functional Block Diagram of Reset Controller During Power-On Reset



The CC_RESET_N is generated on the assertion of PO_RESET_N. This is a power-on reset signal to the fabric alignment clock controller (FACC).

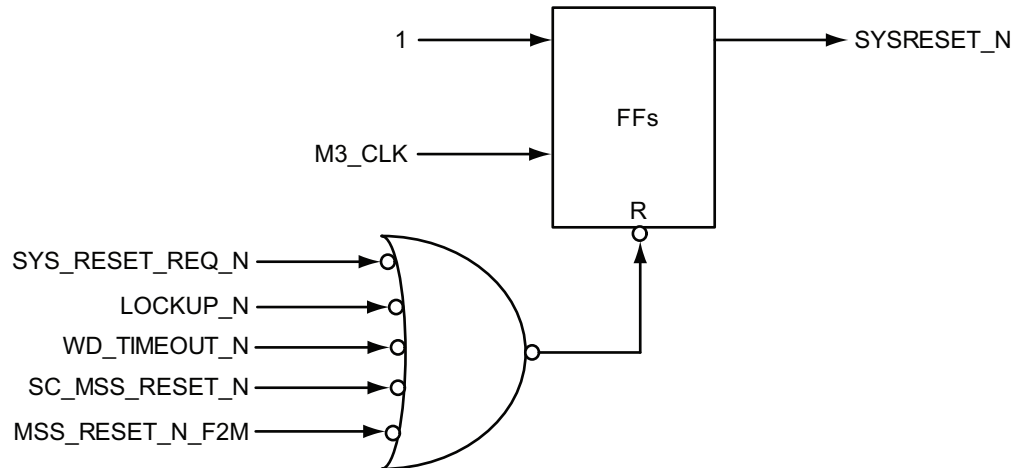
21.2.5 System Reset

The system reset (SYSRESET_N) is generated if any of the following conditions are true:

- SYS_RESET_REQ is asserted from Cortex-M3 processor. SYS_RESET_REQ from the Cortex-M3 processor is controlled by the SYSRESETREQ bit in the Application Interrupt and the Reset Control register located at 0XE000ED0C. For more information, refer to [Cortex-M3 Processor \(Reference Material\)](#), page 18.
- LOCKUP_N is asserted from Cortex-M3 processor in the LOCKUP state. The processor enters into LOCKUP state, if a fault occurs when executing the NMI or HardFault handlers.
- Watchdog timeout event from the Watchdog Timer.
- SC_MSS_RESET_N is asserted from the System Controller during the start-up sequence after power-up.
- MSS_RESET_N_F2M is asserted from the FPGA fabric interface.

The generation of SYSRESET_N is shown in the following figure.

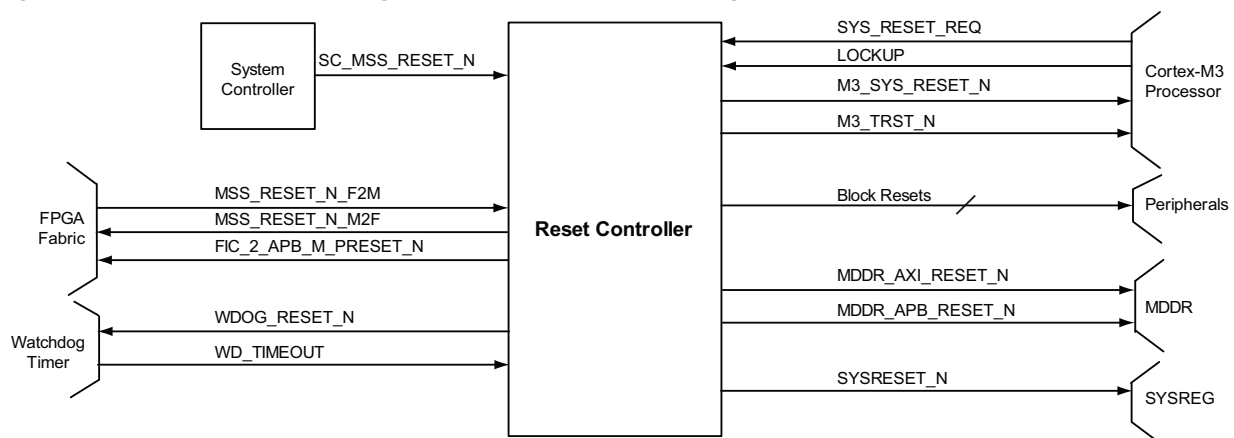
Figure 286 • SYSRESET_N Generation



The inputs SYS_RESET_REQ_N, LOCKUP_N, WD_TIMEOUT_N, SC_MSS_RESET_N, and MSS_RESET_N_F2M are first synchronized on M3_CLK and then combined. The MSS_RESET_N_F2M signal can be used to reset the MSS, independently of any resets coming from the MSS itself. For example it may be asserted as a result of an external reset event from an off-chip Reset Controller, using an I/O pad to bring the reset input into the fabric.

The following figure shows the various reset signals to the MSS blocks which are generated from Reset Controller on assertion of SYSRESET_N. It also shows the reset inputs to the Reset Controller, which cause the generation of SYSRESET_N.

Figure 287 • Functional Block Diagram of Reset Controller During SYSRESET_N



SYSRESET_N resets all blocks in the MSS. When SYSRESET_N asserts low, the entire Cortex-M3 processor is reset, except for the debug logic that exists in the following blocks:

- Nested vectored interrupt controller (NVIC)
- Flash patch and breakpoint (FPB)
- Data watchpoint and trace (DWT)

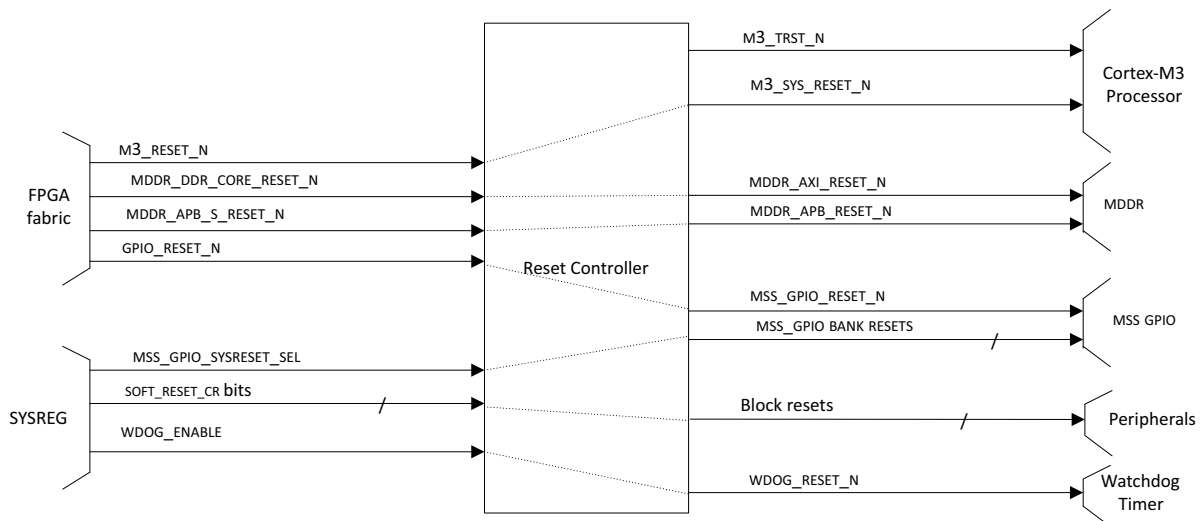
- Instrumentation trace macrocell (ITM)
- AHB-AP

21.2.6 Block Resets

The Reset Controller generates block level resets for all modules except the AHB bus matrix, cache controller, fabric interface interrupt controller (FIIC), RTC, and SYSREG. These blocks will be reset at power-on reset or system reset. The Reset Controller receives block enable bits and soft reset requests (SOFT_RESET_CR bits) for various blocks within the MSS from SYSREG to control the block level reset generation. The Reset Controller also generates four GPIO resets which will reset each bank of GPIO signals, as described in the [MSS GPIO Bank Resets Generator](#), page 659.

The following figure shows the block level resets from the Reset Controller along with the source of the resets.

Figure 288 • Reset Controller With Only Block Level Resets



21.2.6.1 Cortex-M3 Processor Resets

The Reset Controller drives various reset signals to the Cortex-M3 processor as listed below:

1. M3_PORESET_M3_CLK_N
2. M3_PORESET_TCK_N
3. T_RESET_N
4. M3_SYS_RESET_N
5. M3_TRST_N

The source of the Cortex-M3 processor reset captured in the [RESET_SOURCE_CR](#) register (defined in [Table 648](#), page 669). The register captures the status of the resets so that once the Cortex-M3 processor comes out of reset, it can read this register and take further necessary action.

Cortex-M3 processor reset signal generation is explained in the following sections.

21.2.6.1.1 M3_PORESET_M3_CLK_N

This signal resets all logic within the Cortex-M3 processor except the debug logic. M3_PORESET_M3_CLK_N is a synchronized signal of PO_RESET_N on M3_CLK. It asserts asynchronously and negates synchronously to M3_CLK.

21.2.6.1.2 M3_PORESET_TCK_N

This is a reset signal to the Cortex-M3 processor. M3_PORESET_TCK_N is a synchronized signal of PO_RESET_N on TCK (JTAG clock from the clock controller). It asserts asynchronously and negates synchronously to TCK.

21.2.6.1.3 T_RESET_N

This is also a reset signal to the Cortex-M3 processor. The T_RESET_N is a synchronized signal of PO_RESET_N on TRACECLK (Trace clock input from the clock controller). It asserts asynchronously and negates synchronously to TRACECLK.

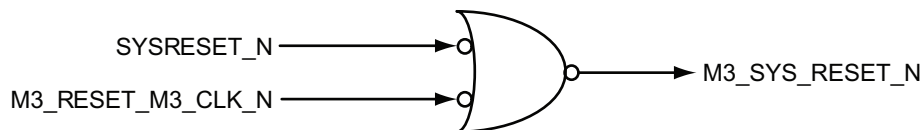
21.2.6.1.4 M3_SYS_RESET_N

M3_SYS_RESET_N resets the Cortex-M3 processor core and its components, excluding the debug logic.

This reset is generated based on M3_RESET_M3_CLK_N and SYSRESET_N.

M3_RESET_M3_CLK_N is a synchronized signal of M3_RESET_N from the FPGA fabric on M3_CLK. The generation of M3_SYS_RESET_N is shown in the following figure.

Figure 289 • M3_SYS_RESET_N Generation



In the five Cortex-M3 processor resets, M3_SYS_RESET_N is the only reset signal that can be controlled.

M3_RESET_N is an active low reset input from the FPGA fabric and resets the Cortex-M3 processors if set to 0. It is only usable in order to extend the duration of system reset to the Cortex-M3 processor after the rest of the MSS has been released from reset. This allows to perform a secure hardware based code shadowing function, thereby minimizing boot time.

21.2.6.1.5 M3_TRST_N

This signal originates from System Controller and drives the NTRST (debug reset) input of the Cortex-M3 processor and is used to reset the SWJ-DP sub-block within the Cortex-M3 processor.

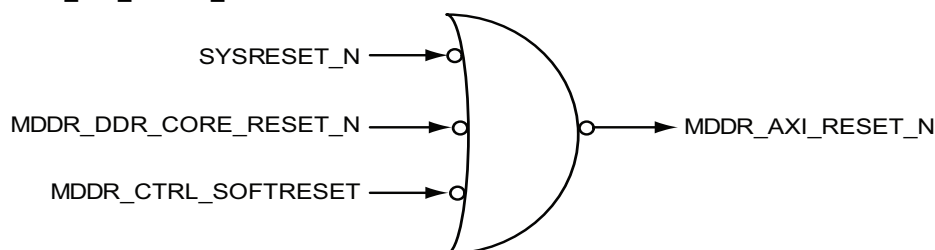
21.2.6.2 MDDR Resets

21.2.6.2.1 MDDR_AXI_RESET_N

MDDR_AXI_RESET_N is generated from FPGA fabric reset input (MDDR_DDR_CORE_RESET_N), SYSRESET_N, and the MDDR soft reset (MDDR_CTRL_SOFTRESET) from SYSREG.

The generation of MDDR_AXI_RESET_N is shown in the following figure.

Figure 290 • MDDR_AXI_RESET_N Generation



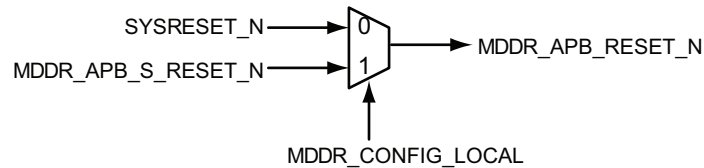
The Reset Controller drives a synchronized reset to AXI logic in the MDDR.

21.2.6.2.2 MDDR_APB_RESET_N

MDDR_APB_RESET_N is generated from FPGA fabric PRESET input (MDDR_APB_S_RESET_N) or SYSRESET_N, based on the selection of MDDR_CONFIG_LOCAL in SYSREG. The MDDR_CONFIG_LOCAL bit is in the MDDR configuration register (MDDR_CR as defined in [Table 648](#), page 669) of SYSREG.

The generation of MDDR_APB_RESET_N is shown in the following figure.

Figure 291 • MDDR_APB_RESET_N Generation



The Reset Controller drives a synchronized reset to the APB logic of the MDDR subsystem.

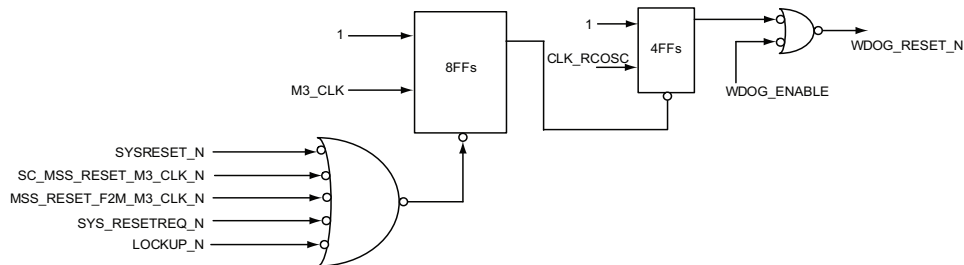
21.2.6.3 Watchdog Resets

21.2.6.3.1 WDOG_RESET_N

The WDOG_BLOCK_RESET_N signal is synchronized on M3_CLK and CLK_RCOSC, then gated with WDOG_ENABLE. The gating ensures that if WDOG_ENABLE is not asserted, WDOG_RESET_N will be asserted. This reset is used to hold the watchdog logic clocked by M3_CLK in reset. The WDOG_ENABLE bit is in the watchdog configuration register (WDOG_CR) as defined in [Table 648](#), page 669) of the SYSREG.

The generation of WDOG_RESET_N is shown in the following figure.

Figure 292 • WDOG_RESET_N Generation



The Reset Controller drives the reset input of the Watchdog Timer.

21.2.6.3.2 PO_RESET_RCOSC_N

The PO_RESET_RCOSC_N is a synchronized version of the PO_RESET_N signal on CLK_RCOSC. It asserts asynchronously and negates synchronously to CLK_RCOSC.

This is a power-on reset signal to the Watchdog Timer.

21.2.6.4 Clock Controller Reset

21.2.6.4.1 CC_RESET_N

The CC_RESET_N is generated on the assertion of PO_RESET_N. This is a power-on reset signal to the fabric alignment clock controller (FACC).

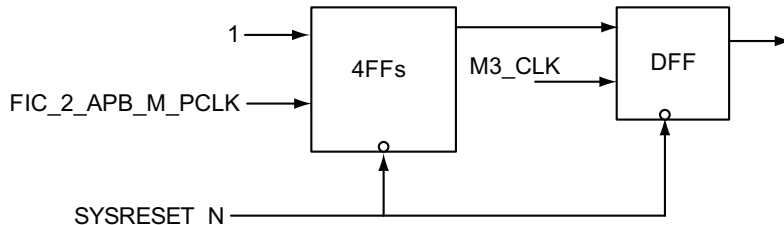
21.2.6.4.2 FIC_2_APB_M_PRESET_N

This is an APB reset signal to the FPGA fabric interface.

SYSRESET_N is synchronized on FIC_2_APB_M_PCLK generated in the MSS CCC, which is driven to the FPGA fabric and then the synchronized reset is again flopped on the negative edge of M3_CLK.

The generation of FIC_2_APB_M_PRESET_N is shown in the following figure.

Figure 293 • FIC_2_APB_M_PRESET_N Generation



21.2.6.5 MSS GPIO Bank Resets Generator

The MSS GPIOs bank can be selectively reset through SYSREG or from flash bits. The GPOUT register is split into four banks of one byte each and each bank has a reset signal.

The Reset Controller receives two SYSREG bits: Soft reset and Select. The Select bit determines the reset source for the associated MSS GPOUT byte. The source can be either of the following:

- Soft reset from system register
- Hard reset derived from either power-on reset or the GPIO_RESET_N signal from the FPGA fabric.

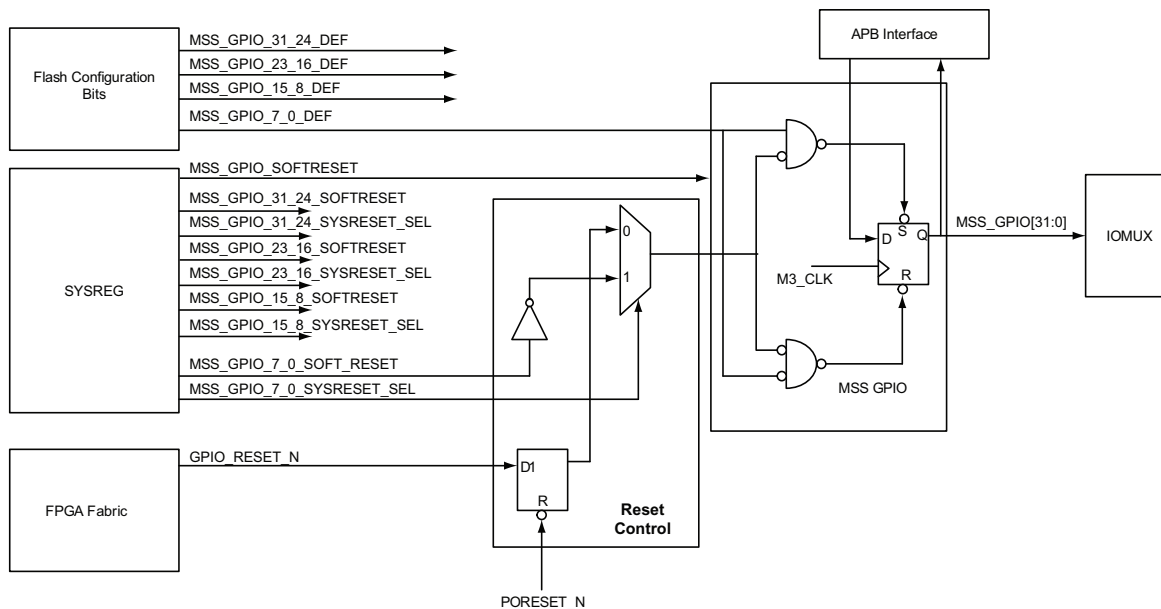
The GPIO_SYSRESET_SEL_CR register (defined in [Table 648](#), page 669) can be configured to select one of the reset inputs. The entire GPIO bank can be kept in reset by asserting the MSS_GPIO_SOFTRESET bit in SOFT_RESET_CR (defined in [Table 648](#), page 669) of SYSREG. A particular GPIO byte can be reset by asserting the corresponding MSS_GPIO_xx_xx_SOFTRESET bit in SOFT_RESET_CR (defined in [Table 648](#), page 669) of SYSREG.

The generation of GPOUT bank resets for flash bit control is shown in the following figure. Only one GPOUT byte bank reset is shown; the resets for other banks are generated in a similar way.

The flash bits to the MSS GPIO are used to initialize the GPOUT byte on assertion of reset. When reset is deasserted, the GPOUT byte will follow the Switch Input (D). When one of the flash bits is deasserted, the associated GPIO_OUT pins are initialized to '0'. The following table explains the generation of GPIO_OUT. MSS_GPIO_xx_xx_DEF bits are in the MSS_GPIO_DEF register of SYSREG.

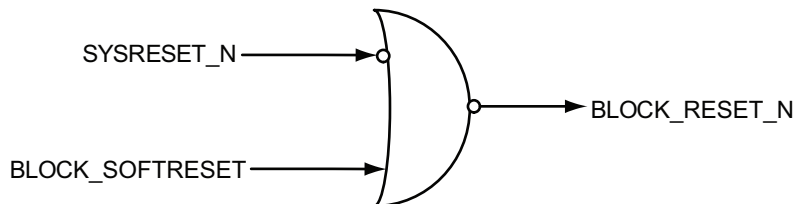
Table 647 • GPIO_OUT Bank Reset Generation

MSS_GPIO_xx_xx_DEF	Reset Controller o/p	GPIO_OUT o/p
0	0	0
0	1	D
1	0	1
1	1	D

Figure 294 • MSS GPIO_OUT Reset Generation

21.2.6.6 Reset Generation to MSS Peripherals

The Reset Controller generates block level resets for the peripherals present within the MSS. The block level reset generation is shown in the following figure.

Figure 295 • Block Level Reset Generation

The reset signal is asserted if any of the following conditions is true:

- SYSRESET_N asserted
- Block level Soft reset (SOFT_RESET_CR) request asserted from SYSREG module.

The Reset Controller can generate the reset to ENV_M_0, ENV_M_1(if present), ESRAM_0, ESRAM_1, Ethernet MAC, PDMA, MMUART_0, MMUART_1, SPI_0, SPI_1, I2C_0, I2C_1, TIMER, CAN (if present), HPDMA, USB OTG, COMM_BLK, FIC_0, FIC_1 (if present), MSS_GPIO (MSS_GPIO_RESET_N reset), and the FPGA fabric (MSS_RESET_N_M2F reset).

21.3 CoreResetP Soft Reset Controller

The following Reset sub-systems in SmartFusion2 devices that must be sequenced properly for the overall system to function correctly.

- Chip Boot (System Controller)
- Fabric
- MSS, Cortex-M3 processor
- FIC sub-systems (MSS to Fabric and Fabric to MSS)
- Peripherals - MDDR, FDDR and SERDESIF

CoreResetP Soft Reset Controller gathers various reset signals from system controller, MSS and FPGA fabric and generates new synchronized reset signals to handles the sequencing of reset signals of various subsystems in SmartFusion2 devices. CoreResetP helps manage the following:

- The FIC sub-systems resets: Both MSS and FPGA fabric should be out of reset to establish the communication between them. CoreResetP generates MSS_READY signal which indicates that both MSS and FPGA fabric are out of reset and ready for communication.
- The Peripherals Initialization: It generates resets signals to initialize MDDR, FDDR and SERDESIF peripheral blocks.
- The Peripherals Reconfiguration: Individual reset controls via CoreConfigP Soft core.
- The PCIe L2/P2 (in-band) and PRST# (out-band) low power modes for all devices, except M2S090.

21.3.1 Reset Topology

This section describes the following reset topology which needs to be applied to the user design.

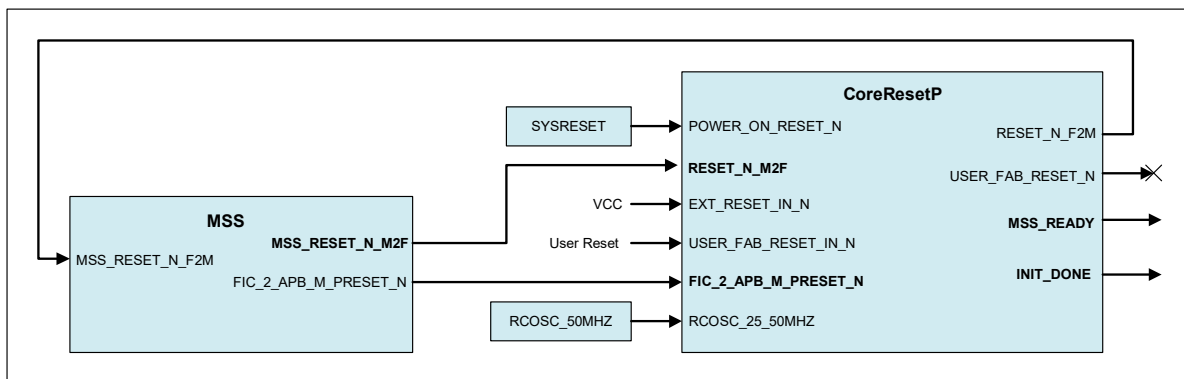
- MSS_READY Generation
- Peripheral Initialization
- SERDES L2/P2, PRST#

21.3.1.1 MSS_READY Generation

Any design which consists of MSS and a fabric subsystem must be synchronized to establish the communication between them. When MSS is doing any transaction, the fabric should be ready. Similarly when fabric is doing any transaction, the MSS should be ready. The following figure shows the typical FIC subsystem with CoreResetP connectivity. CoreResetP generates MSS_READY signal which indicates that MSS is ready for communication. MSS_READY signal is generated whenever a cold reset (power-up event or assertion of DEVRST_N) occurs or due to MSS reset (e.g. watchdog timeout event, assertion of MSS_RESET_N_F2M etc). CoreResetP is relying on MSS_RESET_N_M2F and FIC_2_APB_M_PRESET_N signal to generate MSS_READY signal.

If the user logic consists of any of the two DDR controllers (FDDR or MDDR) or Serial High speed controller (SERDESIF), the INIT_DONE signal should be used to reset the fabric subsystem. If none of MDDR/FDDR/SERDES is used, the MSS_READY signal should be used to reset the fabric subsystem. If the System Builder is used to generate the Libero project, all required cores are Instantiated, and connections are made automatically.

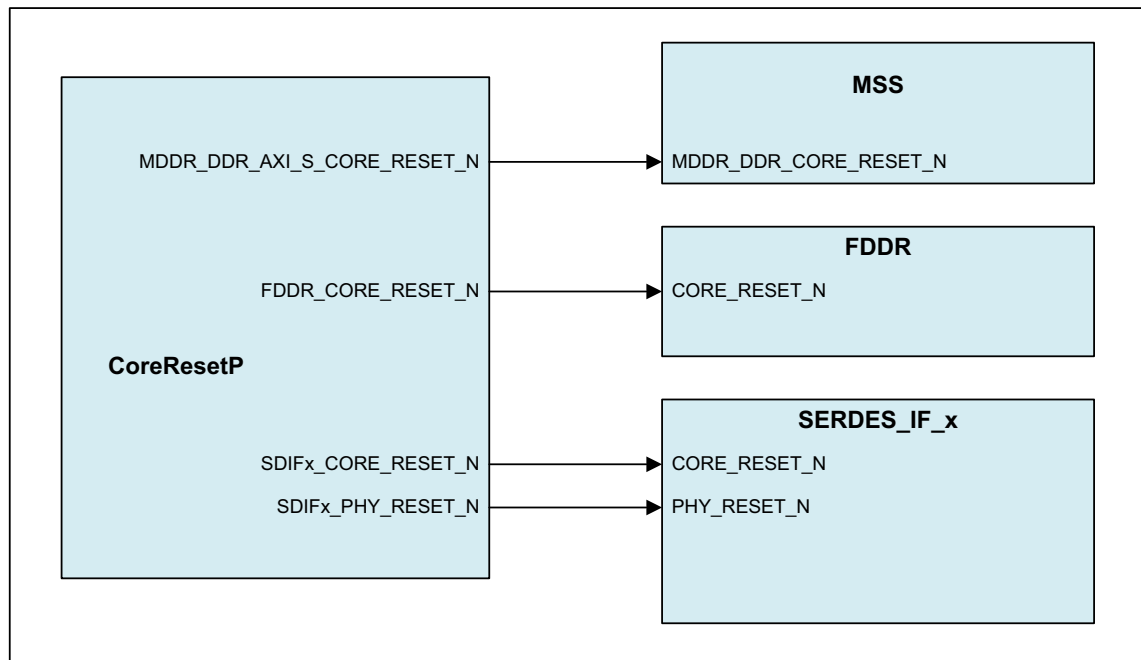
Figure 296 • MSS_READY Signal Generation



21.3.1.2 Peripheral Initialization

CoreResetP generates reset signals to initialize MDDR, FDDR and SERDES_IF peripheral blocks. The following figure shows the CoreResetP connectivity with peripheral resets. For each SERDES_IF blocks, the CoreResetP generates SDIFx_PHY_RESET_N and SDIFx_CORE_RESET_N signals which need to be connected to SERDES_IF macro on PHY_RESET_N and CORE_RESET_N respectively. For FDDR and MDDR, the CoreResetP generates CORE reset signals (FDDR_CORE_RESET_N and MDDR_DDR_AXI_S_CORE_RESET_N).

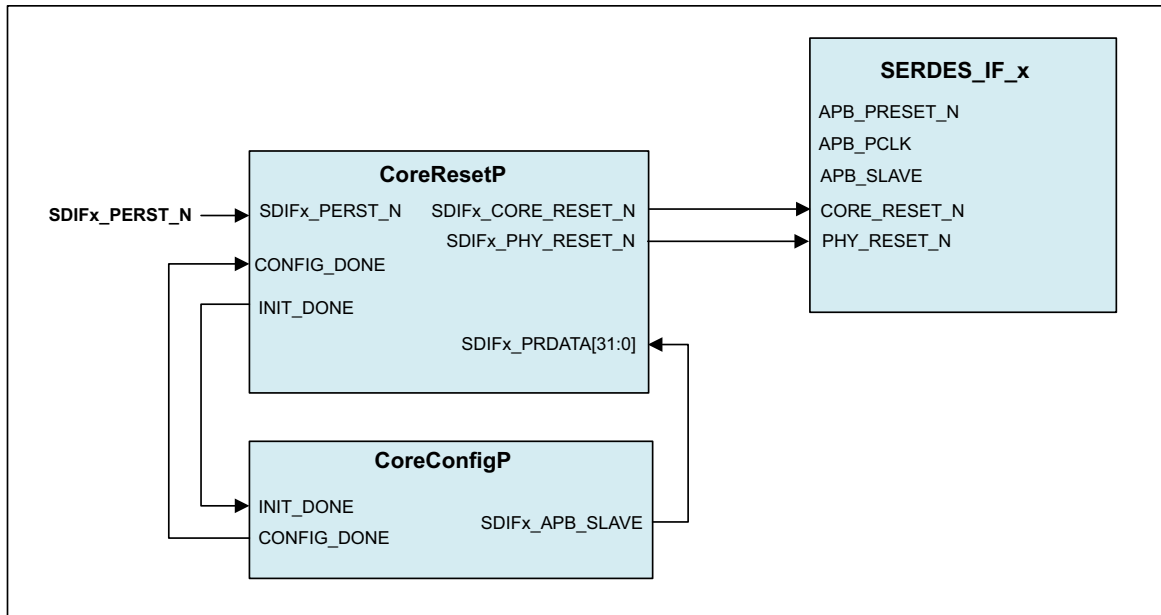
Figure 297 • CoreResetP Connectivity with Peripheral Resets



21.3.1.3 SERDES L2/P2, PRST#

L2 and P2 are low power states for the Link and PHY interface in a PCI Express (PCIe) system. A power management component in a PCIe system will control exit from the L2/P2 state. Part of the sequence when emerging from the low power state involves assertion and release of the PCI Express Reset (PERST# or SDIFx_PERST_N in our implementation). CoreResetP monitors SDIFx_PERST signals and L2/P2 state and generates CORE reset and PHY reset to fulfill the low power mode reset requirement. The following figure shows the CoreResetP connectivity with SERDES_IF block. If the System Builder is used to generate the Libero project, all required cores are Instantiated, and connections are made automatically.

Figure 298 • CoreResetP Connectivity with SERDES_IF Block



21.3.2 Implementation

If the System Builder tool is used within the Libero SoC software to construct a design targeted at a SmartFusion2 device, CoreResetP will automatically be instantiated and connected within the design if required. You can manually instantiate and configure CoreResetP within a SmartDesign design if required. Refer to the CoreResetP Handbook for connecting and configuring CoreResetP in SmartDesign.

Note: CoreConfigP soft IP facilitates configuration of peripheral blocks (MDDR, FDDR, and SERDESIF blocks) in a SmartFusion2 devices. CoreConfigP is available in the Libero SoC IP Catalog. Refer to the CoreConfigP Handbook for port lists and their descriptions, design flows, memory maps, and Control and Status register details.

21.3.3 Timing Diagrams

The following figures show the timing of reset signals for reset sequences initiated by the assertion of POWER_ON_RESET_N, FIC_2_APB_M_PRESET_N, EXT_RESET_IN_N, and USER_FAB_RESET_IN_N signals.

Figure 299 • Timing for Reset Signals Initiated by the Assertion of POWER_N_RESET_N

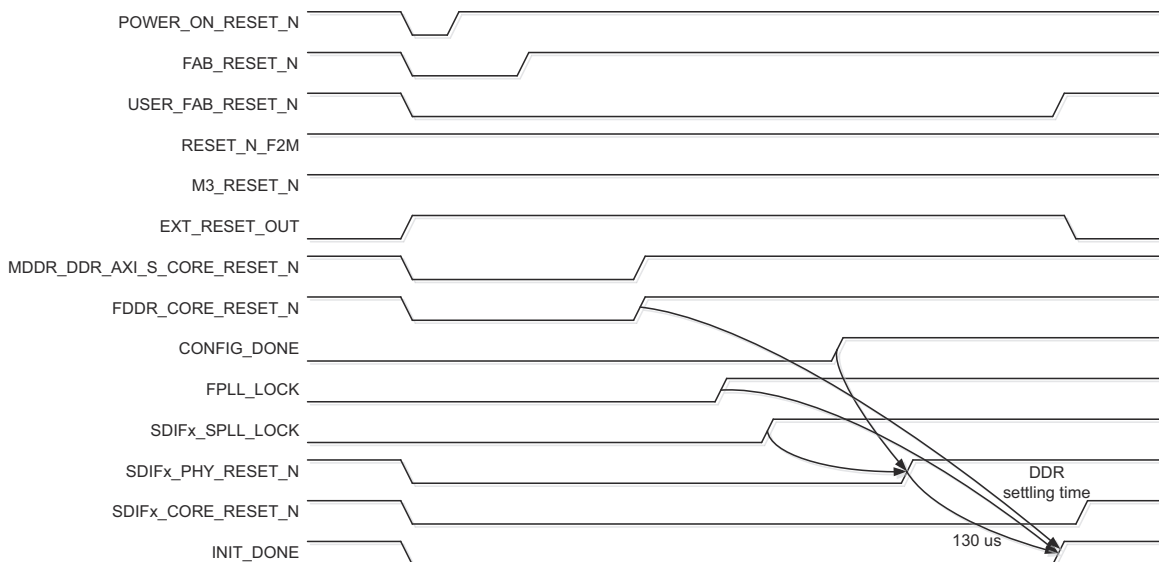


Figure 300 • Timing for Reset Signals Initiated by the Assertion of FIC_2_APB_M_PRESET_N

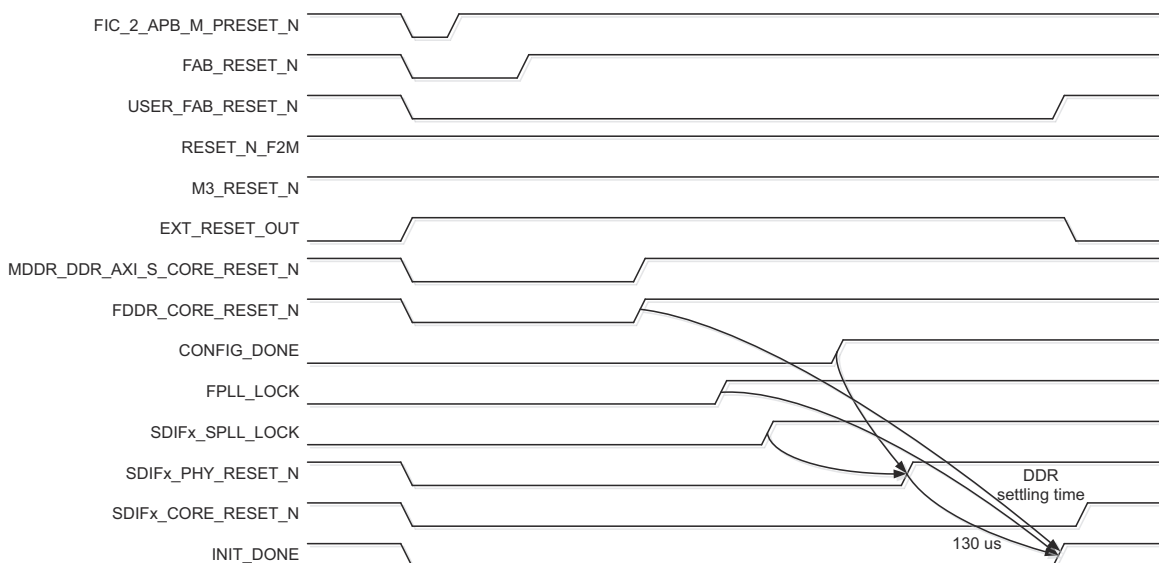
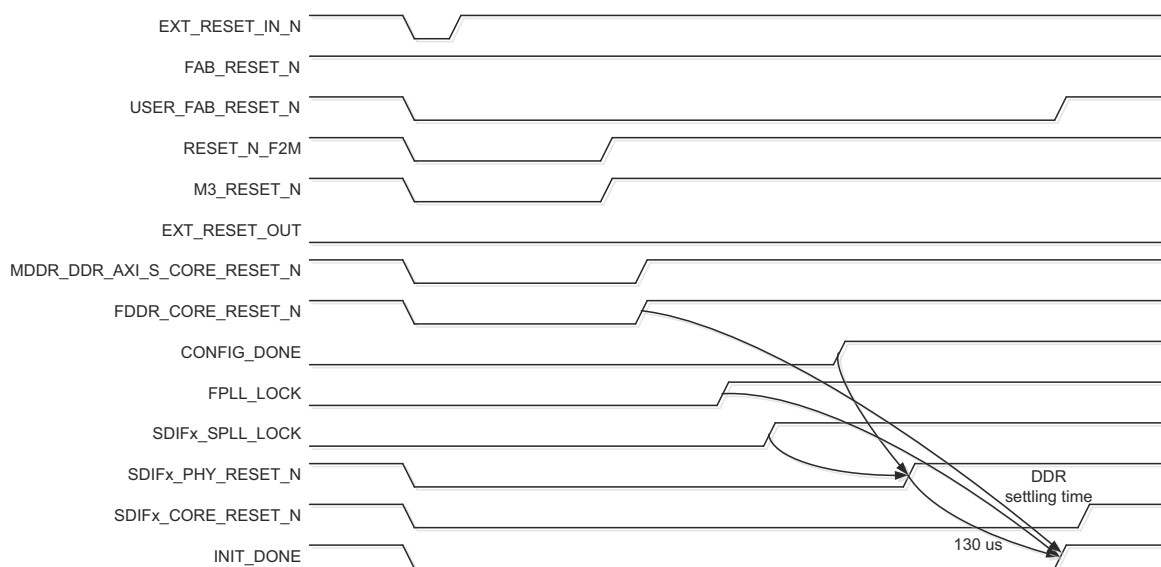


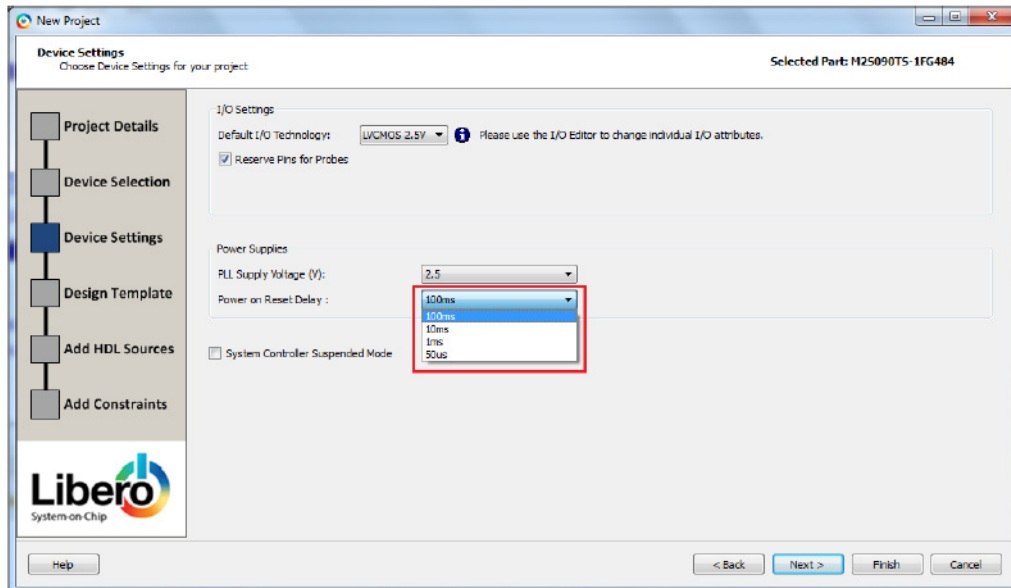
Figure 301 • Timing for Reset Signals Initiated by the Assertion of EXT_RESET_IN_N

Figure 302 • Timing for Reset Signals Initiated by the Assertion of USER_FAB_RESET_IN_N


21.4 How to Use the Reset Controller

21.4.1 Ramp Delay Configuration

The delay can be configured to 50 μ s, 1 ms, 10 ms, or 100 ms in the **New Project** window while creating the Libero SoC project as shown in the following figure. Users can also access and change this setting after the project has been created from the **Project Settings** window.

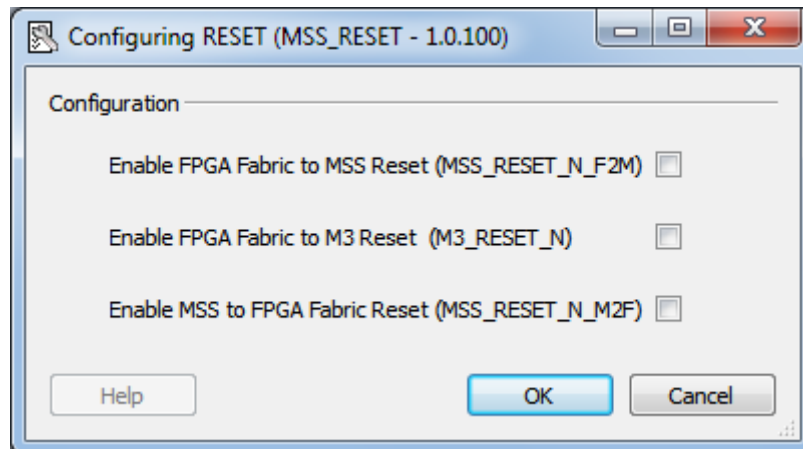
Figure 303 • Ramp Delay Configuration



21.4.2 Reset Controller Configurator

The resets `MSS_RESET_N_F2M`, `M3_RESET_N`, and `MSS_RESET_N_M2F` can be enabled using the Reset Controller configurator in Libero SoC, as shown in the following figure.

Figure 304 • Configuring Reset



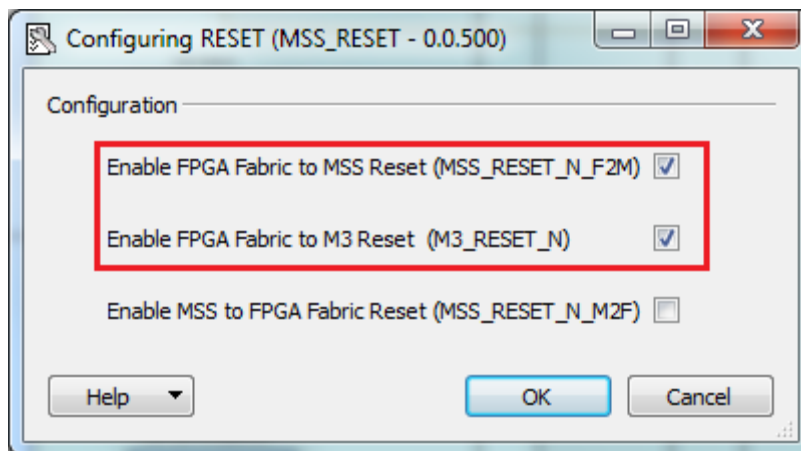
21.4.2.1 Use Model 1: Resetting Cortex- M3 Processor from Fabric

M3_RESET_N is used to hold the Cortex-M3 processor in a reset state after MSS reset.

Use the following steps for holding the Cortex-M3 reset from the fabric.

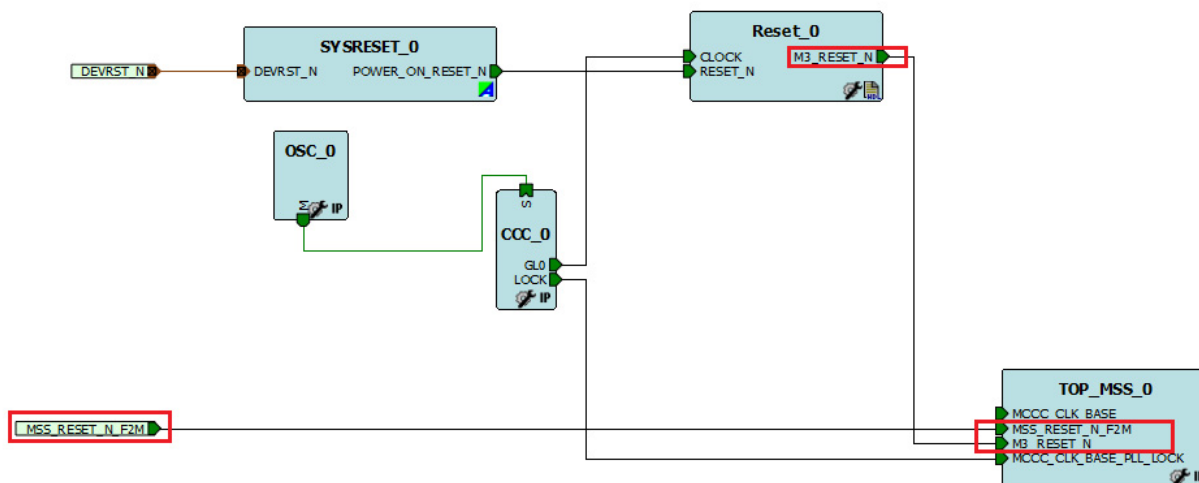
1. Instantiate the SmartFusion2 MSS component in the SmartDesign canvas.
2. Configure the SmartFusion2 MSS peripheral components as needed using the MSS configurator.
3. Configure the Reset Controller, as shown the following figure.

Figure 305 • Configuring Reset



4. Instantiate the fabric logic in the SmartDesign canvas. Connect the fabric logic to M3_RESET_N.
5. Instantiate the fabric CCC and SYSRESET_N for driving the clock and reset to fabric logic.
6. Connect the fabric logic to M3_RESET_N and make the other connections, as shown in the following figure. MSS_RESET_N_F2M is promoted to the top level for resetting the MSS from an external switch.

Figure 306 • Connecting Fabric Logic



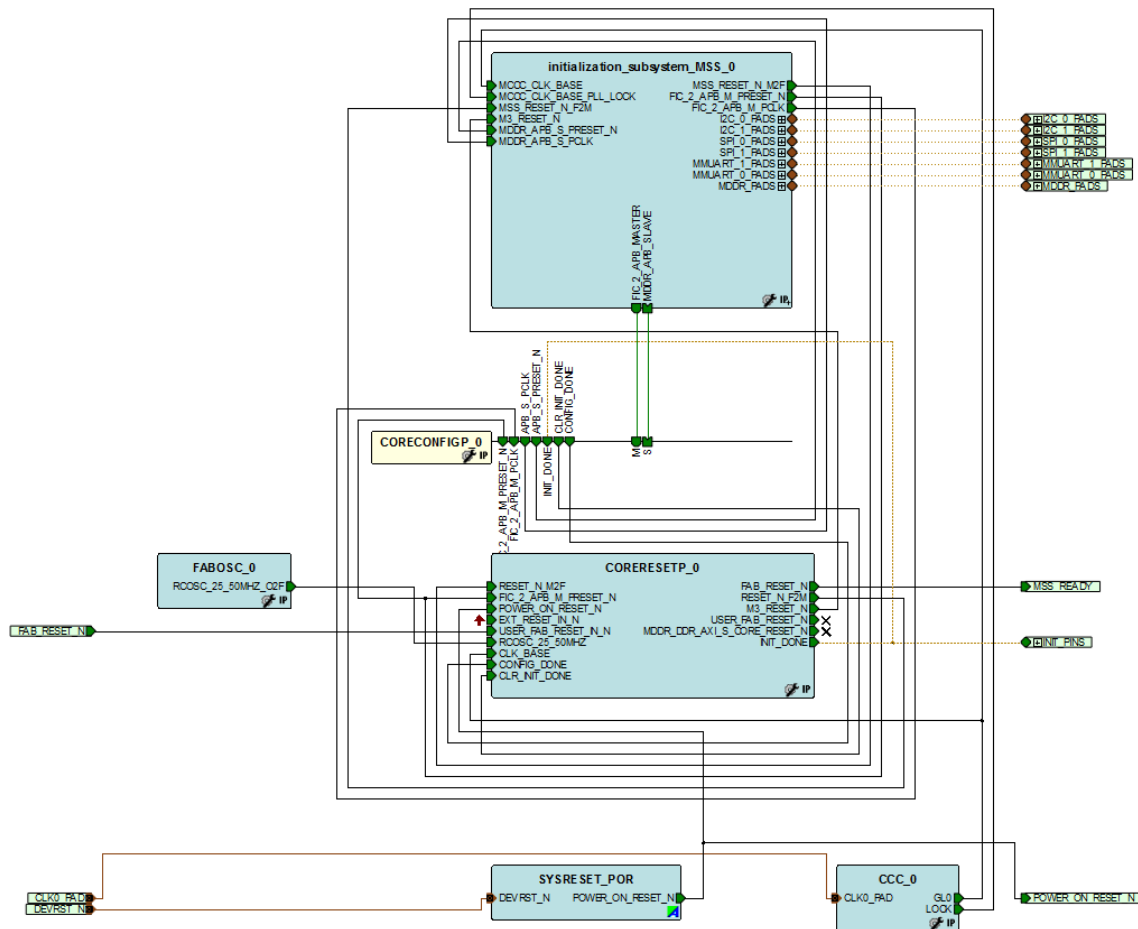
21.4.2.2 Use Model 2: Creating Initialization Sub-system for MDDR

Use the following steps for creating Initialization Sub-system using the System Builder. CoreResetP, CoreConfigP, SYSRESET and Oscillator are Instantiated and connections are made automatically.

1. Select **Use System Builder** while creating a new project from the Design Templates and Creators panel in Libero SoC.
2. Check MSS External Memory in the System Builder - Device Features GUI.

- Follow the rest of the steps with default settings and generate the design.
The following figure shows the generated design when opened in SmartDesign.
The actual SmartDesign created by System Builder is typically not visible unless you open the generated output as a SmartDesign. Refer to the [System Builder User Guide](#) for more details.

Figure 307 • Initialization Sub-system with CoreResetP Soft IP



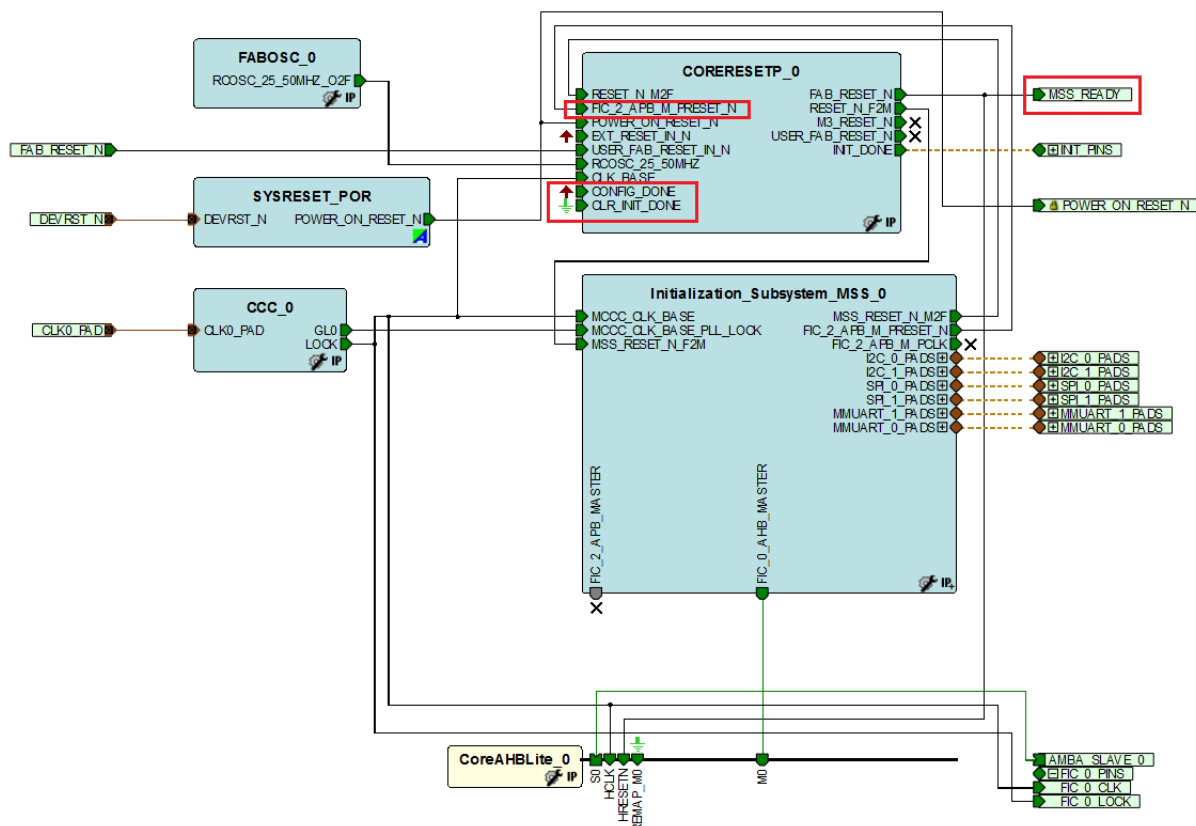
21.4.2.3 Use Model 3: Creating Initialization Sub-system for FIC Sub-systems

Use the following steps for creating Initialization Sub-system using the System Builder. CoreResetP, SYSRESET and Oscillator are Instantiated and connections are made automatically.

- Select **Use System Builder** while creating a new project from the Design Templates and Creators panel in Libero SoC.
- System builder - Device Features GUI is pop up. Click **Next**.
- Drag **Fabric AMBA Slave** core from Fabric Slave Cores panel into MSS FIC_0 – MSS Master Subsystem.
- Follow the rest of the steps with default settings and generate the design.

The following figure shows the generated design when opened in SmartDesign.

The actual SmartDesign created by System Builder is typically not visible unless you open the generated output as a SmartDesign. Refer to the [System Builder User Guide](#) for more details.

Figure 308 • Initialization Sub-system for FIC Sub-systems

21.5 SYSREG Control Registers

The description of registers are located in the SYSREG section of the user's guide and are listed in the following table. Refer to the [System Register Block](#), page 670 for a detailed description of each register and bit.

Table 648 • Switch Register Map

Register Name	Register Type	Flash Write Protect	Reset Source	Description
GPIO_SYSRESET_SEL_CR	RW-P	Register	PORESET_N	Configures the GPIO system reset
SOFT_RESET_CR	RW-P	Bit	SYSRESET_N	Generates the software control resets to the MSS peripherals
RESET_SOURCE_CR	RW			Reset source control register. The source of Cortex-M3 processor reset is captured in this register. The reset values are mentioned in the bit definitions.
MDDR_CR	RW-P	Register	PORESET_N	MDDR configuration register
WDOG_CR	RW-P	Register	PORESET_N	It configures Watchdog timer
MSSDDR_FACC1_CR	RW-P	Field	CC_RESET_N	MSS DDR Bridge fabric alignment clock controller 1 configuration register

22 System Register Block

The System Register (SYSREG) block is an array of system-level registers that contain user configuration information used to configure the microcontroller subsystem (MSS). The contents of these registers are initially set based on the information entered using the MSS configurator in the Libero software. The power-up initialized state of these registers, as well as write protection bits are controlled by flash configuration bits. The configuration bits are set during device programming.

The SYSREG block is connected to the AHB bus matrix and can be accessed by all bus masters. Write access to these registers provides the capability to modify the initialized SYSREG block register contents by the user application. There are seven types of System Registers as described in [Table 649](#), page 671 which provide different levels of read/write access by bus masters.

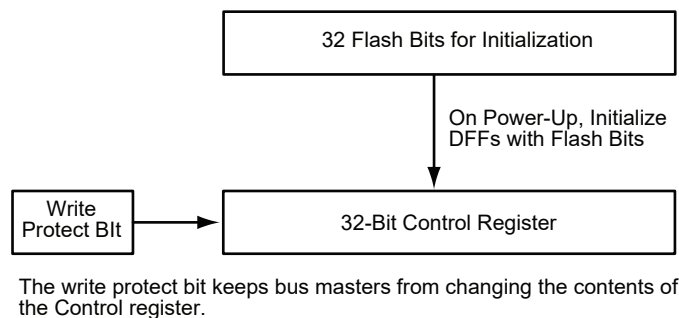
22.0.1 SYSREG Block Register Write Protection

Each SYSREG block register has dedicated write protect bits to control write access from bus masters. Write protect bits are flash configuration bits that are set based on user inputs to the MSS configurator. These bits are defined during the device design phase and can only be modified by reprogramming the device. Users have the ability to set protection levels for the entire register, independent fields within each register, or individual bits within each register.

22.0.1.1 Register Write Protect

One Register Write Protect bit is used to write protect entire register contents as shown in the following figure. On power-up, the register contents are initialized based on the flash configuration bits set from the MSS configurator. If the Register Write Protect bit is set in the MSS configurator, the initialized value of the entire register cannot be modified by the user application. If the Register Write Protect bit is not set, the contents of the register can be modified by any bus master. Register Write Protect bits can only be modified by reprogramming the FPGA and is therefore protected by the standard FPGA programming security features.

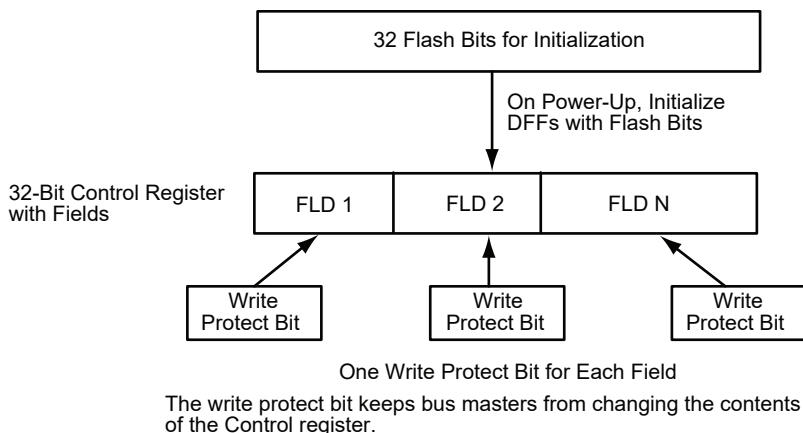
Figure 309 • Register Write Protect



22.0.1.2 Field Write Protect

Many System Registers contain fields of multiple bits. A Field Write Protect bit provides write protection for an entire field within a single register as shown in the following figure. Field Write Protect bits follow the same rules as Register Write Protect bits described in [Register Write Protect](#), page 670, but are instead applied to each individual field within the register. There is a Field Write Protect bit allocated for each field within the register.

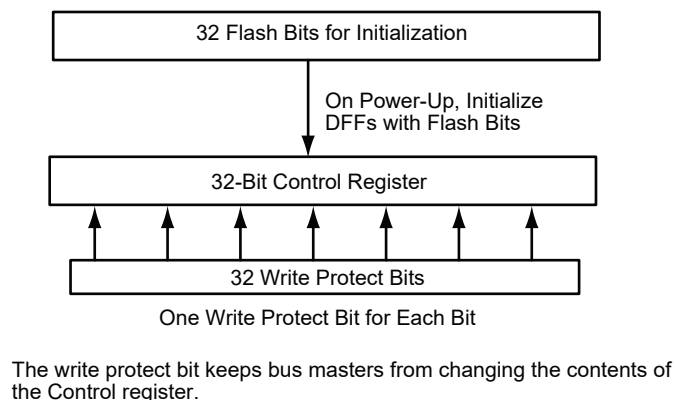
Figure 310 • Field Write Protect



22.0.1.3 Bit Write Protect

A Bit Write Protect bit provides write protection for each individual bit within a single register as shown in the following figure. Bit Write Protect bits follow the same rules as Register Write Protect bits described in [Register Write Protect](#), page 670, but are instead applied to each individual bit within the register. There is a Bit Write Protect bit allocated for each bit within the register.

Figure 311 • Bit Write Protect



Refer to the [Register Lock Bits Configuration](#), page 674 for locking and unlocking registers.

22.0.2 Register Types

There are several register types in the SYSREG block as defined in the following table.

Table 649 • Register Types

Type	Function
RW-P	Supports read and write accesses via AHB bus matrix. Refer to Figure 312 , page 672. Register contents are initialized from flash configuration bits at power-up and the assertion of SYS_RESET_N. Typically used for MSS Control Registers.

Table 649 • Register Types (continued)

Type	Function
RW	Supports read and write accesses via AHB bus matrix. Refer to Figure 313 , page 673. Register contents are not initialized from flash configuration bits at power-up. The reset state is determined by the user HW design following assertion of SYS_RESET_N. Typically used for MSS Control Registers.
RO	Supports read only accesses via AHB bus matrix. Refer to Figure 314 , page 673. Register contents are not initialized from flash configuration bits at power-up or the assertion of SYS_RESET_N. Typically used for MSS Control Registers.
RO-U	Does not support read or write access via AHB bus matrix. Refer to Figure 316 , page 674. Register contents are initialized from flash configuration bits at power-up and the assertion of SYS_RESET_N. Typically used for MSS Control Registers.
RO-P	Supports read only accesses via AHB bus matrix. Refer to Figure 315 , page 674. Register contents are initialized from flash configuration bits at power-up and the assertion of SYS_RESET_N. Typically used to return MSS status information.
W1P	Write '1' to clear the register. This register is Write-Only.
SW1C	Individual register bits are set ('1') when related input is asserted. Bits are individually cleared when corresponding register bit is written high.

The following figures explain schematically a few of the register types of the SYSREG registers.

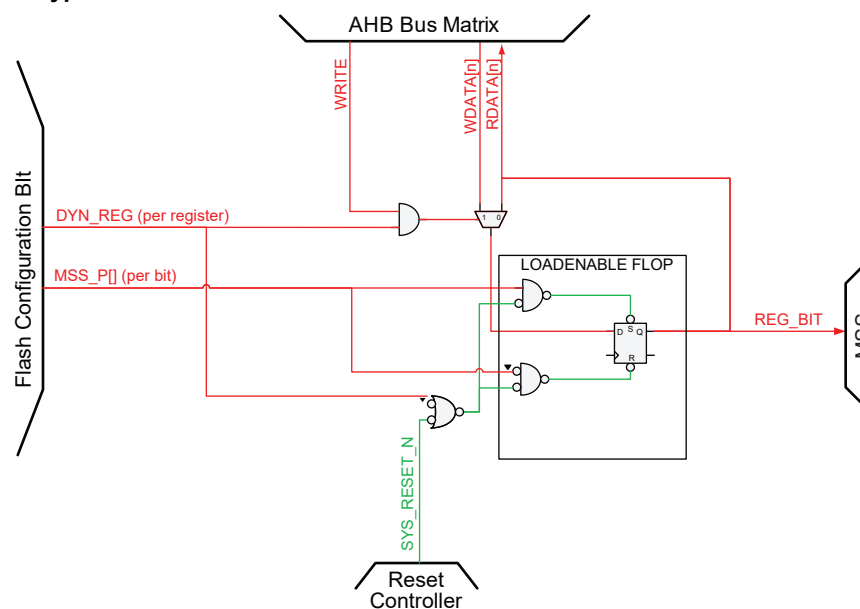
Figure 312 • RW-P Type

Figure 313 • RW Type

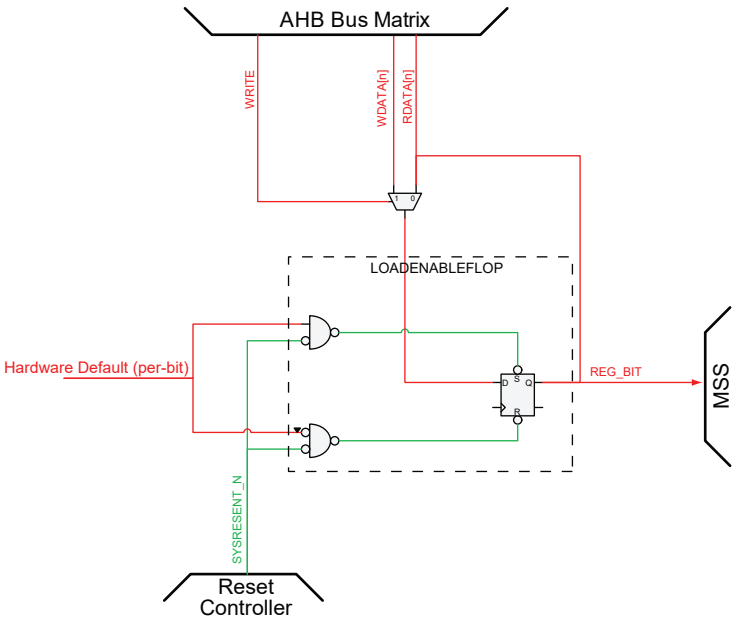


Figure 314 • RO Type

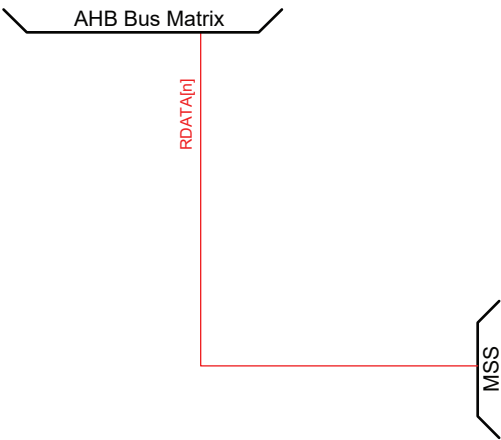
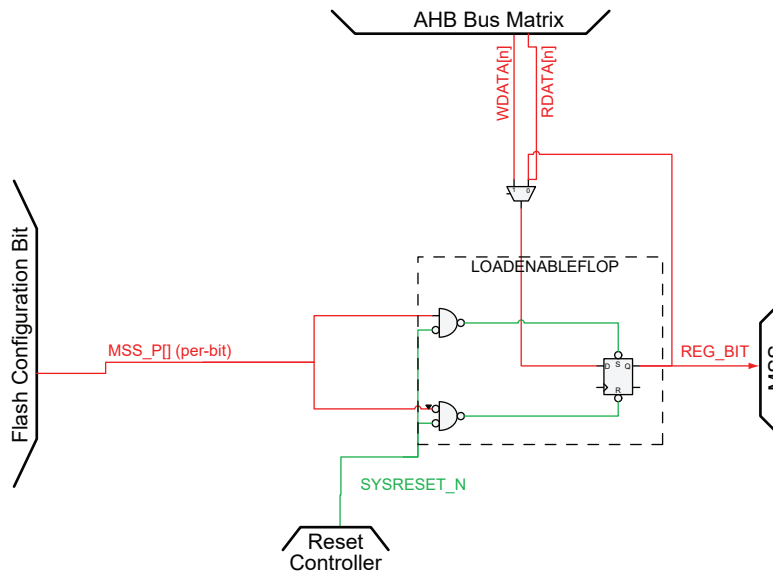
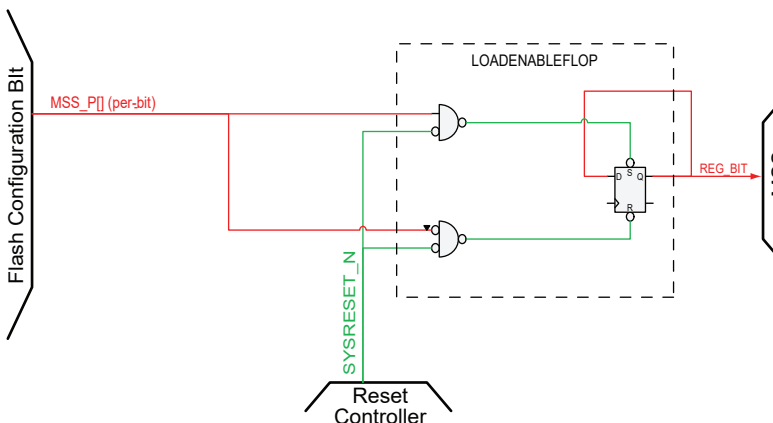
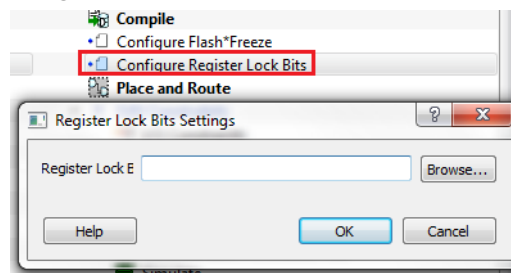


Figure 315 • RO-P Type**Figure 316 • RO-U Type**

22.1 Register Lock Bits Configuration

The Register Lock Bits Configuration tool is used to lock MSS, SERDES, and FDDR configuration registers of SmartFusion2 devices in order to prevent them from being overwritten by masters that have access to these registers. Register lock bits are set in a text (*.txt) file, which is then imported into the SmartFusion2 project. From the **Design Flow** window, click **Configure Register Lock Bits** to open the configurator. Then, click **Browse...** to navigate to the text file (*.txt) that contains the Register Lock Bit settings. (see the following figure).

Figure 317 • Register Lock Bit Settings

22.1.1 Lock Bit File

An initial, default lock bit file can be generated by clicking **Generate FPGA Array Data** in the **Design Flow** window.

The default file located at <proj_location>/designer/<root>/<root>_init_config_lock_bits.txt can be used to make the required changes.

Note: Save the file using a different name if you modify the text file to set the lock bits.

22.1.2 Lock Bit File Syntax

A valid entry in the lock bit configuration file is defined as a <lock_parameters> <lock bit value> pair format.

The lock parameters are structured as follows:

- Lock bits syntax for a register: <Physical block name>_<register name>_LOCK
- Lock bits syntax for a specific field: <Physical block name>_<register name>_<field name>_LOCK
- The following are the physical block names (varies with device family and die):
- MSS
- FDDR
- SERDES_IF_x (where x is 0,1,2,3 to indicate the physical SERDES location) for SmartFusion2 M2S010/025/050/150 devices
- SERDES_IF2 for SmartFusion2 M2S060/090 devices (only one SERDES block per device)

Set the lock bit value to 1 to indicate that the register can be written to (unlocked) and to 0 to indicate that the register cannot be written to (locked).

Lines starting with # or ; are comments. Empty lines are allowed in the lock bit configuration file.

The following figure shows the lock bit configuration file.

Figure 318 • Lock Bit Configuration File

```
# Register Lock Bits Configuration File for MSS, SERDES(s) and Fabric DDR
# Microsemi Corporation - Microsemi Libero Software Release v11.7 SP1 (Version 11.7.1.2)
# Date: Tue Mar 29 13:24:54 2016

# sb_sb_0/sb_sb_MSS_0/MSS_ADLIB_INST/INST_MSS_050_IP
MSS_ESRAM_CONFIG_LOCK      0
MSS_ESRAM_MAX_LAT_LOCK     1
MSS_DDR_CONFIG_LOCK        1
MSS_ENVM_CONFIG_LOCK       0
MSS_ENVM_REMAP_BASE_LOCK   1
MSS_ENVM_FAB_REMAP_LOCK    1
MSS_CC_CONFIG_LOCK         0
MSS_CC_CACHEREGION_LOCK    1
MSS_CC_LOCKBASEADDR_LOCK   1
MSS_CC_FLUSHIDX_LOCK       0
MSS_DDRB_BUF_TIMER_LOCK    1
MSS_DDRB_NB_ADR_LOCK       1
MSS_DDRB_NB_SIZE_LOCK      0
MSS_DDRB_CONFIG_LOCK       1
MSS_EDAC_ENABLE_LOCK       1
MSS_MASTER_WEIGHT_CONFIG0_LOCK 1
MSS_MASTER_WEIGHT_CONFIG1_LOCK 1
MSS_SOFT_INTERRUPT_LOCK    1
MSS_SOFTRESET_ENVM0_SOFTRESET_LOCK 1
MSS_SOFTRESET_ENVM1_SOFTRESET_LOCK 1
MSS_SOFTRESET_ESRAM0_SOFTRESET_LOCK 1
MSS_SOFTRESET_ESRAM1_SOFTRESET_LOCK 1
MSS_SOFTRESET_MAC_SOFTRESET_LOCK 1
MSS_SOFTRESET_PDMA_SOFTRESET_LOCK 1
MSS_SOFTRESET_TIMER_SOFTRESET_LOCK 1
MSS_SOFTRESET_MMUART0_SOFTRESET_LOCK 1
MSS_SOFTRESET_MMUART1_SOFTRESET_LOCK 1
MSS_SOFTRESET_G4SPI0_SOFTRESET_LOCK 1
MSS_SOFTRESET_G4SPI1_SOFTRESET_LOCK 1
MSS_SOFTRESET_I2C0_SOFTRESET_LOCK 1
MSS_SOFTRESET_I2C1_SOFTRESET_LOCK 1
MSS_SOFTRESET_CAN_SOFTRESET_LOCK 1
MSS_SOFTRESET_USB_SOFTRESET_LOCK 1
MSS_SOFTRESET_COMBLK_SOFTRESET_LOCK 1
MSS_SOFTRESET_FPGA_SOFTRESET_LOCK 1
MSS_SOFTRESET_HPDM_SOFTRESET_LOCK 1
MSS_SOFTRESET_FIC32_0_SOFTRESET_LOCK 1
MSS_SOFTRESET_FIC32_1_SOFTRESET_LOCK 1
MSS_SOFTRESET_MSS_GPIO_SOFTRESET_LOCK 1
MSS_SOFTRESET_MSS_GPOUT_7_0_SOFT_RESET_LOCK 1
MSS_SOFTRESET_MSS_GPOUT_15_8_SOFT_RESET_LOCK 1
MSS_SOFTRESET_MSS_GPOUT_23_16_SOFT_RESET_LOCK 1
MSS_SOFTRESET_MSS_GPOUT_31_24_SOFT_RESET_LOCK 1
MSS_SOFTRESET_MDDR_CTLR_SOFTRESET_LOCK 1
MSS_SOFTRESET_MDDR_FIC64_SOFTRESET_LOCK 1
MSS_M3_CONFIG_LOCK        1
```

22.1.3 Locking and Unlocking a Register

A register can be locked or unlocked by setting the appropriate lock bit value in the lock bit configuration .txt file.

1. Browse to locate the lock bit configuration .txt file.
2. Do one or both of the following:
 - Set the lock bit value to 0 for the registers you want to lock.
 - Set the lock bit value to 1 for the registers you want to unlock.
3. Save the file, and import the file into the project (**Design Flow** window > **Configure Register Lock Bits**), see [Figure 317](#), page 674.
4. Regenerate the bitstream.

22.2 Register Map

The following table lists all the registers in the SYSREG block. The SYSREG block is located at address 0x40038000 in Cortex-M3 processor address space.

Table 650 • SYSREG

Register Name	Addr. Offset	Register Type	Flash Write Protect	Reset Source	Description
ESRAM_CR	0x0	RW-P	Register	SYSRESET_N	Controls address mapping of the eSRAMs
ESRAM_MAX_LAT	0x4	RW-P	Register	SYSRESET_N	eSRAM0 and eSRAM1 maximum latency
DDR_CR	0x8	RW-P	Register	SYSRESET_N	DDR Configuration Register
ENVM_CR	0xC	RW-P	Register	SYSRESET_N	eNVM Configuration Register
ENVM_REMAP_BASE_CR	0x10	RW-P	Register	SYSRESET_N	eNVM Remap Base Address Control Register
ENVM_REMAP_FAB_CR	0x14	RW-P	Register	SYSRESET_N	eNVM Remap Base Address Control Register
CC_CR	0x18	RW-P	Register	SYSRESET_N	Used to configure cache
CC_REGION_CR	0x1C	RW-P	Register	SYSRESET_N	Cache Region Control Register
CC_LOCK_BASE_ADDR_CR	0x20	RW-P	Register	SYSRESET_N	Cache Lock Base Address Control Register
CC_FLUSH_INDXX_CR	0x24	RW-P	Register	SYSRESET_N	Cache Flush Index Control Register
DDRB_BUF_TIMER_CR	0x28	RW-P	Register	SYSRESET_N	DDR write buffer timeout
DDRB_NB_ADDR_CR	0x2C	RW-P	Register	SYSRESET_N	DDR non-bufferable address region base address
DDRB_NB_SIZE_CR	0x30	RW-P	Register	SYSRESET_N	Size of non- bufferable address region
DDRB_CR	0x34	RW-P	Register	SYSRESET_N	MSS DDR bridge Configuration Register

Table 650 • SYSREG (continued)

Register Name	Addr. Offset	Register Type	Flash Write Protect	Reset Source	Description
EDAC_CR	0x38	RW-P	Register	SYSRESET_N	EDAC Configuration Register for eSRAM0, eSRAM1, USB, MAC, and CAN
MASTER_WEIGHT0_CR	0x3C	RW-P	Register	SYSRESET_N	Master Weight Configuration Register 0
MASTER_WEIGHT1_CR	0x40	RW-P	Register	SYSRESET_N	Master Weight Configuration Register 1
SOFT_IRQ_CR	0x44	RW-P	Register	SYSRESET_N	Enables software interrupt
SOFT_RESET_CR	0x48	RW-P	Bit	SYSRESET_N	Software Reset Control Register
M3_CR	0x4C	RW-P	Register	SYSRESET_N	Cortex M3 Configuration Register
FAB_IF_CR	0x50	RW-P	Register	SYSRESET_N	Controls fabric interface
LOOPBACK_CR	0x54	RW-P	Register	SYSRESET_N	Controls MSS peripherals
GPIO_SYSRESET_SEL_CR	0x58	RW-P	Register	PORESET_N	Configures GPIO system reset
GPIN_SRC_SEL_CR	0x5C	RW-P	Register	PORESET_N	GPIO Input Source Select Control Register
MDDR_CR	0x60	RW-P	Register	PORESET_N	MDDR Configuration Register
USB_IO_INPUT_SEL_CR	0x64	RW-P	Register	PORESET_N	Configures USB data interfaces from IOMUXCELLs and I/O pads
PERIPH_CLK_MUX_SEL_CR	0x68	RW-P	Register	PORESET_N	Peripheral Clock MUX Select Control Register
WD OG_CR	0x6C	RW-P	Register	PORESET_N	Configures Watchdog timer
MDDR_IO_CALIB_CR	0x70	RW-P	Register	PORESET_N	MDDR I/O Calibration Control Register
Reserved	0x74				
EDAC_IRQ_ENABLE_CR	0x78	RW-P	Register	SYSRESET_N	Enables/disables 1-bit error, 2-bit error status for eSRAM0, eSRAM1, USB, CAN, and MAC
USB_CR	0x7C	RW-P	Register	SYSRESET_N	Configures USB interface
ESRAM_PIPELINE_CR	0x80	RW-P	Register	SYSRESET_N	Controls the pipeline present in the memory read path of eSRAM memory
MSS_IRQ_ENABLE_CR	0x84	RW-P	Register	SYSRESET_N	MSS Interrupt Enable Control Register
RTC_WAKEUP_CR	0x88	RW-P	Register	SYSRESET_N	Configures RTC timer WAKEUP signal

Table 650 • SYSREG (continued)

Register Name	Addr. Offset	Register Type	Flash Write Protect	Reset Source	Description
MAC_CR	0x8C	RW-P	Register	SYSRESET_N	MAC Configuration Register
MSSDDR_PLL_STATUS_LOW_CR	0x90	RW-P	Register	CC_RESET_N	Controls the configuration input of MPLL register
MSSDDR_PLL_STATUS_HIGH_CR	0x94	RW-P	Register	CC_RESET_N	Controls the configuration input of MPLL register
MSSDDR_FACC1_CR	0x98	RW-P	Field	CC_RESET_N	MSS DDR bridge FACC1 Configuration Register
MSSDDR_FACC2_CR	0x9C	RW-P	Field	CC_RESET_N	MSS DDR bridge FACC2 Configuration Register
PLL_LOCK_EN_CR	0xA0	RW-P	Register	CC_RESET_N	PPL Lock Enable Control Register
MSSDDR_CLK_CALIB_CR	0xA4	RW-P	Register	SYSRESET_N	Starts FPGA fabric calibration test circuit
PLL_DELAY_LINE_SEL_CR	0xA8	RW-P	Register	SYSRESET_N	PLL Delay Line Select Control Register
MAC_STAT_CLRONRD_CR	0xAC	RW-P	Register	SYSRESET_N	MAC status clear on read
RESET_SOURCE_CR	0xB0	RW			Reset Source Control Register
CC_DC_ERR_ADDR_SR	0xB4	RO		SYSRESET_N	Dcode Bus Error Address Status Register
CC_IC_ERR_ADDR_SR	0xB8	RO		SYSRESET_N	Icode Bus Error Address Status Register
CC_SB_ERR_ADDR_SR	0xBC	RO		SYSRESET_N	System Bus Error Address Status Register
Reserved	0xC0			SYSRESET_N	
CC_IC_MISS_CNTR_SR	0xC4	RO		SYSRESET_N	ICode Miss Control Status Register
CC_IC_HIT_CNTR_SR	0xC8	RO		SYSRESET_N	ICode Hit Control Status Register
CC_DC_MISS_CNTR_SR	0xCC	RO		SYSRESET_N	DCode Miss Control Status Register
CC_DC_HIT_CNTR_CR	0xD0	RO		SYSRESET_N	DCode Hit Control Status Register
CC_IC_TRANS_CNTR_SR	0xD4	RO		SYSRESET_N	ICode Transaction Count Control Status Register
CC_DC_TRANS_CNTR_SR	0xD8	RO		SYSRESET_N	DCode Transaction count Control Status Register
DDRB_DS_ERR_ADR_SR	0xDC	RO		SYSRESET_N	MSS DDR Bridge DS Master Error Address Status Register

Table 650 • SYSREG (continued)

Register Name	Addr. Offset	Register Type	Flash Write Protect	Reset Source	Description
DDRB_HPD_ERR_ADR_SR	0xE0	RO		SYSRESET_N	MSS DDR Bridge High Performance DMA Master Error Address Status Register
DDRB_SW_ERR_ADR_SR	0xE4	RO		SYSRESET_N	MSS DDR Bridge AHB Bus Error Address Status Register
DDRB_BUF_EMPTY_SR	0xE8	RO		SYSRESET_N	MSS DDR Bridge Buffer Empty Status Register
DDRB_DSBL_DN_SR	0xEC	RO		SYSRESET_N	MSS DDR Bridge Disable Buffer Status Register
ESRAM0_EDAC_CNT	0xF0	RO		SYSRESET_N	1-bit error and 2-bit error count of eSRAM0
ESRAM1_EDAC_CNT	0xF4	RO		SYSRESET_N	1-bit error and 2-bit error count of eSRAM1
Reserved	0xF8			SYSRESET_N	
MAC_EDAC_TX_CNT	0xFC	RO		SYSRESET_N	1-bit error and 2-bit error count of MAC transmitter
MAC_EDAC_RX_CNT	0x100	RO		SYSRESET_N	1-bit error and 2-bit error count of MAC receiver
USB_EDAC_CNT	0x104	RO		SYSRESET_N	1-bit error and 2-bit error count of USB
CAN_EDAC_CNT	0x108	RO		SYSRESET_N	1-bit error and 2-bit error count of CAN
ESRAM0_EDAC_ADR	0x10C	RO		SYSRESET_N	Address from eSRAM0 on which 1-bit and 2-bit SECEDED error has occurred
ESRAM1_EDAC_ADR	0x110	RO		SYSRESET_N	Address from eSRAM1 on which 1-bit and 2-bit SECEDED error has occurred
MAC_EDAC_RX_ADR	0x114	RO		SYSRESET_N	Address from MAC receiver on which 1-bit and 2-bit SECEDED error has occurred
MAC_EDAC_TX_ADR	0x118	RO		SYSRESET_N	Address from MAC transmitter on which 1-bit and 2 bit SECEDED error has occurred.
CAN_EDAC_ADR	0x11C	RO		SYSRESET_N	Address from CAN on which 1-bit and 2-bit SECEDED error has occurred
USB_EDAC_ADR	0x120	RO		SYSRESET_N	Address from USB on which 1-bit and 2-bit SECEDED error has occurred
MM0_1_2_SECURITY	0x124	RO-U		SYSRESET_N	Read and write security for masters 0, 1, and 2 to eSRAM0, eSRAM1, eNVM1, eNVM0, and MSS DDR bridge

Table 650 • SYSREG (continued)

Register Name	Addr. Offset	Register Type	Flash Write Protect	Reset Source	Description
MM4_5_DDR_FIC_SECURITY / MM4_5_FIC64_SECURITY	0x128	RO-U		SYSRESET_N	Read and write security for masters 4, 5, and DDR_FIC to eSRAM0, eSRAM1, eNVM1, eNVM0, and MSS DDR bridge
MM3_6_7_8_SECURITY	0x12C	RO-U		SYSRESET_N	Read and write security for masters 3, 6, 7, and 8 to eSRAM0, eSRAM1, eNVM1, eNVM0, and MSS DDR bridge
MM9_SECURITY	0x130	RO-U		SYSRESET_N	Read and write security for master 9 to eSRAM0, eSRAM1, eNVM1, eNVM0, and MSS DDR bridge
M3_SR	0x134	RO		SYSRESET_N	Cortex-M3 processor Status Register
ETM_COUNT_LOW	0x138	RO		SYSRESET_N	ETM count for lower bits [31:0]
ETM_COUNT_HIGH	0x13C	RO		SYSRESET_N	ETM count for higher bits [47:32]
DEVICE_SR	0x140	RO		SYSRESET_N	Device Status Register
ENVM_PROTECT_USER	0x144	RO-U		SYSRESET_N	Configuration for accessibility of protect regions of eNVM0 and eNVM1
ENVM_STATUS	0x148	RO-U		PORESET_N	Code shadow Status Register
DEVICE_VERSION	0x14C	RO			Configures device version
MSSDDR_PLL_STATUS	0x150	RO			MSS DDR PLL Status Register
USB_SR	0x154	RO		SYSRESET_N	USB Status Register
ENVM_SR	0x158	RO		SYSRESET_N	Busy status eNVM0 and eNVM1
Reserved	0x15C				
DDRB_STATUS	0x160	RO		SYSRESET_N	MSS DDR bridges status
MDDR_IO_CALIB_STATUS	0x164	RO		PORESET_N	DDR I/O Calibration Status Register
MSSDDR_CLK_CALIB_STATUS	0x168	RO		SYSRESET_N	MSS DDR Clock Calibration Status Register
WDGLOAD	0x16C	RO-P		PORESET_N	Configures Watchdog load value
WDGMVRP	0x170	RO-P		PORESET_N	Configures Watchdog MVRP value
USERCONFIG0	0x174	RO-P		SYSRESET_N	User Configuration Register 0

Table 650 • SYSREG (continued)

Register Name	Addr. Offset	Register Type	Flash Write Protect	Reset Source	Description
USERCONFIG1	0x178	RO-P		SYSRESET_N	User Configuration Register 1
USERCONFIG2	0x17C	RO-P		SYSRESET_N	User Configuration Register 2
USERCONFIG3	0x180	RO-P		SYSRESET_N	User Configuration Register 3
FAB_PROT_SIZE	0x184	RO-P		SYSRESET_N	Size of memory protected from fabric master
FAB_PROT_BASE	0x188	RO-P		SYSRESET_N	Base address which is protected from fabric master
MSS_GPIO_DEF	0x18C	RO-P		SYSRESET_N	MSS GPIO Definition Register
EDAC_SR	0x190	SW1C		SYSRESET_N	Status of 1-bit SECEDED error detection and correction, 2-bit SECEDED error detection for eSRAM0, eSRAM1, MAC, USB, and CAN
MSS_INTERNAL_SR	0x194	SW1C		SYSRESET_N	MSS Internal Status Register
MSS_EXTERNAL_SR	0x198	SW1C		SYSRESET_N	MSS External Status Register
WDOGTIMEOUTEVENT	0x19C	SW1C		PORESET_N	Watchdog Time out event register
CLR_MSS_COUNTERS	0x1A0	W1P		SYSRESET_N	Clear MSS counters
CLR_EDAC_COUNTERS	0x1A4	W1P		SYSRESET_N	Clears 16-bit counter value in eSRAM0, eSRAM1, MAC, USB, and CAN corresponding to count value of EDAC 1-bit and 2-bit errors
FLUSH_CR	0x1A8	W1P		SYSRESET_N	Flush Control Register
MAC_STAT_CLR_CR	0x1AC	W1P		SYSRESET_N	MAC Statistics Clear Control Register
IOMUXCELL_CONFIG[n] n is 0 to 56	0x1B0 to 0x290	RW-P	Register	PORESET_N	I/O MUXCELL Configuration Register

22.3 Register Details

22.3.1 System Registers Behavior for M2S005/010 Devices

Application traffic across the FIC_0 interface can cause certain bits in the SYSREG block to change state, if these bits are dynamically modified from their default values during runtime. This impacts all SmartFusion2 005 and 010 devices.

The following table lists the subset of system registers and specific bit definitions that are affected. The registers/bits listed in the following table should be configured once on power-up. Dynamically altering the contents of these registers can result in their values to be reset to the power on reset state.

Table 651 • Subset of System Registers

System Register	Fields
SOFT_RESET_CR	All bits
M3_CR	STCALIB
FAB_IF_CR	FAB0_AHB_BYPASS, FAB1_AHB_BYPASS, FAB0_AHB_MODE, FAB1_AHB_MODE, and SW_FIC_REG_SEL
LOOPBACK_CR	MSS_MMUARTLOOPBACK, MSS_SPILOOPBACK, MSS_I2CLOOPBACK, and MSS_GPIOLOOPBACK
GPIO_SYSRESET_SEL_CR	MSS_GPIO_7_0_SYSRESET_SEL, MSS_GPIO_15_8_SYSRESET_SEL, MSS_GPIO_23_16_SYSRESET_SEL, and MSS_GPIO_31_24_SYSRESET_SEL
GPIN_SRC_SEL_CR	MSS_GPINSOURCE
MDDR_CR	MDDR_CONFIG_LOCAL, SDR_MODE, F_AXI_AHB_MODE, and PHY_SELF_REF_EN
USB_IO_INPUT_SEL_CR	USB_IO_INPUT_SEL
PERIPH_CLK_MUX_SEL_CR	SPI0_SCK_FAB_SEL, SPI1_SCK_FAB_SEL, and TRACECLK_DIV2_SEL
WDOG_CR	WDOGENABLE and WDOGMODE
EDAC_IRQ_ENABLE_CR	All bits
MSS_IRQ_ENABLE_CR	DDRB_INTERRUPT_EN, SW_INTERRUPT_EN, and CC_INTERRUPT_EN
RTC_WAKEUP_CR	RTC_WAKEUP_M3_EN, RTC_WAKEUP_FAB_EN, and RTC_WAKEUP_C_EN
MSSDDR_PLL_STATUS_LOW_CR	FACC_PLL_DIVR, FACC_PLL_DIVF, FACC_PLL_DIVQ, FACC_PLL_RANGE, FACC_PLL_LOCKWIN, and FACC_PLL_LOCKCNT
MSSDDR_PLL_STATUS_HIGH_CR	FACC_PLL_BYPASS, FACC_PLL_MODE_1V2, FACC_PLL_MODE_3V3, FACC_PLL_FSE, FACC_PLL_PD, FACC_PLL_SSE, FACC_PLL_SSMD, and FACC_PLL_SSMF
MSSDDR_FACC1_CR	DIVISOR_A, APB0_DIVISOR, APB1_DIVISOR, DDR_CLK_EN, M3_CLK_DIVISOR, FACC_GLMUX_SEL, FIC_0_DIVISOR, and FIC_1_DIVISOR
IOMUXCELL_CONFIG[n]	MSS_IOMUXSEL0[N], MSS_IOMUXSEL1[N], MSS_IOMUXSEL2[N], MSS_IOMUXSEL3[N], MSS_IOMUXSEL4[N][2:0], MSS_IOMUXSEL5MID[N], and MSS_IOMUXSEL5LOWER[N]

22.3.2 eSRAM Configuration Register

Table 652 • ESRAM_CR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	
1	SW_CC_ESRAM1FWREMAP	0	Defines the locations of eSRAM_0 and eSRAM_1 if eSRAM remap is enabled (if SW_CC_ESRAMFWREMAP is asserted). If SW_CC_ESRAMFWREMAP is 0, this bit has no meaning. If SW_CC_ESRAMFWREMAP is 1, this bit has the following definition: 0: eSRAM_0 is located at address 0x00000000 in the ICODE/DCODE space of Cortex-M3 processor and eSRAM_1 is located just above eSRAM_0 (adjacent to it). 1: eSRAM_1 is located at address 0x00000000 in ICODE/DCODE space of Cortex-M3 processor and eSRAM_0 is located just above eSRAM_1 (adjacent to it).
0	SW_CC_ESRAMFWREMAP	0	Indicates that eSRAM_0 and eSRAM_1 are remapped to ICODE/DCODE space of the Cortex-M3 processor. If this bit is 1 and SW_CC_ESRAM1FWREMAP is 0, then eSRAM_0 is at location 0x00000000 and eSRAM_1 is always remapped to be just above eSRAM_0 (the two eSRAMs are adjacent in ICODE/DCODE space). Both eSRAMs also remain visible in SYSTEM space of the Cortex-M3 processor and remain visible at this location to all other (non-Cortex-M3 processor) masters. The bit definitions: 0: No eSRAM remap is enabled. This means that eNVM (or MDDR) is present at location 0x00000000. 1: eSRAM_0 and eSRAM_1 are remapped to location 0x00000000 of Cortex-M3 processor ICODE/DCODE space.

22.3.3 eSRAM Latency Configuration Register

Table 653 • ESRAM_MAX_LAT

Bit Number	Name	Reset Value	Description
[31:6]	Reserved	0	
[5:3]	SW_MAX_LAT_ESRAM1	0x1	Defines the maximum number of cycles the processor bus waits for eSRAM1 when it is being accessed by a master with a WRR priority scheme. The latency values are as given in Table 654 , page 684.
[2:0]	SW_MAX_LAT_ESRAM0	0x1	Defines the maximum number of cycles the processor bus waits for eSRAM0 when it is being accessed by a master with a WRR priority scheme. It is configurable from 1 to 8 (8 by default). The latency values are as given in Table 654 , page 684.

The following table lists eSRAM maximum latency values, where x is either 0 or 1.

Table 654 • eSRAM Maximum Latency Values

SW_MAX_LAT_ESRAM<X>	Latency
0	8 (default)
1	1
2	2
3	3
4	4
5	5
6	6
7	7

22.3.4 DDR Configuration Register

Table 655 • DDR_CR

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	SW_CC_DDRFWREMAP	0	Indicates that DDR_Space0 and DDR_Space1 are remapped to the ICODE/DCODE space of the Cortex-M3 processor. Both DDR spaces also remain visible in the SYSTEM space of the Cortex-M3 processor and remain visible at this location to all other non-Cortex-M3 processor masters. The bit definitions: 0: No DDR space remap is enabled. This means that eNVM is present at location 0x00000000. 1: DDR_Space0 and DDR_Space1 are remapped to location 0x00000000 of Cortex-M3 processor ICODE/DCODE space.

22.3.5 eNVM Configuration Register

Table 656 • ENVM_CR

Bit Number	Name	Reset Value	Description
[31:17]	Reserved	0	
16	ENVM_SENSE_ON	0	Turns on or off the sense amps for both NVM0 and NVM1
15	ENVM_PERSIST	0	Reset control for NVM0 and NVM1 0: Reset on SYSRESET_N and PORESET_N 1: Reset on PORESET_N
14	NV_DPD1	0	Deep power-down control for the NVM1 0: Normal operation 1: NVM deep power-down
13	NV_DPD0	0	Deep power-down control for the NVM0 0: Normal operation 1: NVM deep power-down

Table 656 • ENVM_CR (continued)

[12:5]	NV_FREQRNG	0x7	<p>Setting of NV_FREQRNG[8:5] or NV_FREQRNG[12:9] determines the behavior of eNVM BUSY_B with respect to the AHB Bus interface clock. It can be used to accommodate various frequencies of the external interface clock, M3_CLK, or it can be used to advance or delay the data capture due to variation of read access time of the NVM core. It sets the number of wait states to match with the Cortex-M3 or Fabric master operating frequency for read operations. The small counter in the NVM Controller uses this value to advance or delay the data capture before sampling data.</p> <p>0000: NOT SUPPORTED 0001: NOT SUPPORTED 0010: Page Read = 3, All other modes (Page program and Page verify) = 2 0011: Page Read = 4, All other modes (Page program and Page verify) = 2 0100: Page Read = 5, All other modes (Page program and Page verify) = 2 0101: Page Read = 6, All other modes (Page program and Page verify) = 3 0110: Page Read = 7, All other modes (Page program and Page verify) = 3 0111: Page Read = 8, All other modes (Page program and Page verify) = 4 1000: Page Read = 9, All other modes (Page program and Page verify) = 4 1001: Page Read = 10, All other modes (Page program and Page verify) = 4 1010: Page Read = 11, All other modes (Page program and Page verify) = 5 1011: Page Read = 12, All other modes (Page program and Page verify) = 5 1100: Page Read = 13, All other modes (Page program and Page verify) = 6 1101: Page Read = 14, All other modes (Page program and Page verify) = 6 1110: Page Read = 15, All other modes (Page program and Page verify) = 6 1111: Page Read = 16, All other modes (Page program and Page verify) = 7 NV_FREQRNG[8:5] is used for NVM0 and NV_FREQRNG[12:9] is used for NVM1.</p>
4:0	SW_ENVMREMAPSIZE	0x11	<p>Size of the segment in eNVM which is to be remapped to location 0x00000000. This logically splits eNVM into a number of segments, each of which may be used to store a different firmware image, for example. The region sizes are shown in Table 657, page 686.</p>

22.3.5.1 SW_ENVMREMAPSIZE Bit Combinations

Table 657 • SW_ENVMREMAPSIZE

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Remap Size
0	0	0	0	0	Reserved
0	0	0	0	1	Reserved
0	0	0	1	0	Reserved
0	0	0	1	1	Reserved
0	0	1	0	0	Reserved
0	0	1	0	1	Reserved
0	0	1	1	0	Reserved
0	0	1	1	1	Reserved
0	1	0	0	0	Reserved
0	1	0	0	1	Reserved
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	Reserved
0	1	1	0	1	16 KB
0	1	1	1	0	32 KB
0	1	1	1	1	64 KB
1	0	0	0	0	128 KB
1	0	0	0	1	256 KB
1	0	0	1	0	512 KB, reset value

22.3.6 eNVM Remap Base Address Control Register

Table 658 • ENVM_REMAP_BASE_CR

Bit Number	Name	Reset Value	Description
[31:19]	Reserved	0	
[18:1]	SW_ENVMREMAPBASE	0	Offset within eNVM address space of the base address of the segment in eNVM, which is to be remapped to location 0x00000000. If an eNVM protected region is defined to be read-accessible by the Cortex-M3, then it is read-accessible by Cortex-M3 at both physical and re-mapped addresses. However, if a protected region is defined as writeable by Cortex-M3, then it is writeable via the physical address, but not via the re-mapped address. Bit 0 of this register is defined as SW_ENVMREMAPENABLE. Bit 0 must be set to get the remapping done with new addresses filled in this register.
0	SW_ENVMREMAPENABLE	0	0: eNVM remap not enabled. Bottom of eNVM is mapped to address 0x00000000. 1: eNVM remap enabled. eNVM visible at 0x00000000 is a remapped segment of the eNVM.

Bits [18:N] of this bus indicate the base address of the remapped segment. The value of N depends on the eNVM remap section size, so that the base address is aligned according to an even multiple of the segment size. The power of 2 size specified by SW_ENVMREMAPSIZE[4:0] (Table 657, page 686) defines how many bits of the base address are used. For example, if the SW_ENVMREMAPSIZE[4:0] is 01111, this corresponds to a segment size of 64 KB. 64 KB is 2 to the power of 16. Therefore the value of N in this case, is 16. So the base address of the region, in this case, is specified by SW_ENVMREMAPSIZE[18:16].

This register should only be written by Cortex-M3 processor firmware using 32-bit accesses. The behavior of the system is undefined if other size accesses are used.

22.3.7 eNVM FPGA Fabric Remap Base Address Register

Table 659 • ENVM_REMAP_FAB_CR

Bit Number	Name	Reset Value	Description
[31:19]	Reserved	0	
[18:1]	SW_ENVMFABREMAPBASE	0	Offset within eNVM address space of the base address of the segment in eNVM, which is to be remapped to location 0x00000000 for use by a soft processor in the FPGA fabric.
0	SW_ENVMFABREMAPENABLE	0	0: eNVM fabric remap not enabled for accesses by fabric master. The portion of eNVM visible in the eNVM window at location 0x00000000 of a soft processor's memory space corresponds to the memory locations at the bottom of eNVM. 1: eNVM fabric remap enabled. The portion of eNVM visible at location 0x00000000 of a soft processor's memory space of is a remapped segment of eNVM.

Bits [18:N] of this bus indicate the base address of the remapped segment. The value of N depends on the eNVM remap section size, so that the base address is aligned according to an even multiple of segment size. The power of 2 size specified by SW_ENVMREMAPSIZE[4:0] (Table 657, page 686) defines how many bits of base address are used. For example, if the SW_ENVMREMAPSIZE[4:0] is 01111, this corresponds to a segment size of 64KB. 64KB is 2 to the power of 16. Therefore the value of N in this case, is 16. So the base address of the region, in this case, is specified by SW_ENVMREMAPSIZE[18:16].

This register should only be written by Cortex-M3 processor firmware using 32-bit accesses. The behavior of the system is undefined if other size accesses are used.

22.3.8 Cache Configuration Register

Table 660 • CC_CR

Bit Number	Name	Reset Value	Description
[31:3]	Reserved	0	
2	CC_CACHE_LOCK	0	Allows the cache lock to be enabled. The allowed values: 0: Disabled 1: Enabled
1	CC_SBUS_WR_MODE	0	Allows debug mode SBUS writes to cache memory to be enabled. The allowed values: 0: Disabled 1: Enabled

Table 660 • CC_CR (continued)

0	CC_CACHE_ENB	0	Allows the cache to be disabled. The allowed values: 0: Disabled 1: Enabled
---	--------------	---	---

22.3.9 Cache Region Control Register

Table 661 • CC_REGION_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	
3:0	CC_CACHE_REGION	0	Defines the cache region size. The bits have the following definitions: Bit 0: First (lower) slot of 128 MB (0–128 MB) Bit 1: Second slot of 128 MB (128–256 MB) Bit 2: Third slot of 128 MB (256–384 MB) Bit 3: Fourth (upper) slot of 128 MB (384–512 MB)

For 128 MB configuration, only one bit out of four will be set to select one of the blocks as cacheable.

For 256 MB configuration, bits [1:0] are set to select the lower 256 MB as cacheable, bits [3:2] are set to select the lower 256 MB as cacheable.

For 512 MB configuration all four bits are set to select entire 512 MB as cacheable.

22.3.10 Cache Lock Base Address Control Register

Table 662 • CC_LOCK_BASE_ADDR_CR

Bit Number	Name	Reset Value	Description
[31:19]	Reserved	0	
[18:0]	CC_LOCK_BASEADD	0	If Cache Lock mode is enabled and if there is an SECDED error while accessing the cache memory, the base address can be used to initiate the transaction to re-read the erroneous data from system memory which is stored in this register.

22.3.11 Cache Flush Index Control Register

Table 663 • CC_FLUSH_INDX_CR

Bit Number	Name	Reset Value	Description
[31:6]	Reserved	0	
[5:0]	CC_FLUSH_INDEX	0	Cache memory index to be flushed or invalidated is stored in this register. For 8 KB and four-way associative memory, INDEX_WIDTH will be 6 bits.

22.3.12 MSS DDR Bridge Buffer Timer Control Register

Table 664 • DDRB_BUF_TIMER_CR

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	
[9:0]	DDRB_TIMER	0x3FF	10-bit timer interface used to configure the timeout register in the write buffer module. Once timer reaches the timeout value, a flush request is generated by the flush controller and if response has been received for previous write request from write arbiter, this request is posted to the write arbiter. This register is common for all buffers. The value in this register will be in terms of number of M3_CLK clocks.

22.3.13 MSS DDR Bridge Non-Bufferable Address Control Register

Table 665 • DDRB_NB_ADDR_CR

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	
[15:0]	DDRB_NB_ADDR	0xA000	Base address of a non-bufferable address region. Bits [15:(N – 1)] of this signal are compared with AHB address [31:(N + 15)] to check whether address is in non-bufferable region. The value of N depends on the non-bufferable region size, so the base address is defined according to DDRB_NB_SZ.

22.3.14 MSS DDR Bridge Non-Bufferable Size Control Register

Table 666 • DDRB_NB_SIZE_CR

Bit Number	Name	Reset Value	Description
[31: 46 4]	Reserved	0	
[3:0]	DDRB_NB_SZ	0x1	Size of non-bufferable address region [(2N – 1x64) KB]. The region sizes are as shown in Table 667 , page 689.

The following table lists the size of the non-bufferable region, given by the formula [(2N – 1x64)] KB.

Table 667 • Non-Bufferable Region

Bit 0	Bit 1	Bit 2	Bit 3	N	Non-Bufferable Region
0	0	0	0	0	No non-bufferable region
0	0	0	1	1	64 KB = (2 ¹ – 1 x 64) KB
0	0	1	0	2	128 KB = (2 ² – 1 x 64) KB
0	0	1	1	3	256 KB = (2 ³ – 1 x 64) KB
0	1	0	0	4	512 KB = (2 ⁴ – 1 x 64) KB
0	1	0	1	5	1 MB = (2 ⁵ – 1 x 64) KB

Table 667 • Non-Bufferable Region (continued)

0	1	1	0	6	2 MB = (26 – 1 x 64) KB
0	1	1	1	7	4 MB = (27 – 1 x 64) KB
1	0	0	0	8	8 MB = (28 – 1 x 64) KB
1	0	0	1	9	16 MB = (29 – 1 x 64) KB
1	0	1	0	10	32 MB = (210 – 1 x 64) KB
1	0	1	1	11	64 MB = (211 – 1 x 64) KB
1	1	0	0	12	128 MB = (212 – 1 x 64) KB
1	1	0	1	13	256 MB = (213 – 1 x 64) KB
1	1	1	0	14	512 MB = (214 – 1 x 64) KB
1	1	1	1	15	1 GB = (215 – 1 x 64) KB: Entire region is non-bufferable

22.3.15 MSS DDR Bridge Configuration Register

Table 668 • DDRB_CR

Bit Number	Name	Reset Value	Description
[31:24]	Reserved	0	
[23:20]	DDR_IDC_MAP	0	Sets the DSG interface to DDR address space mapping mode 0–15.
[19:16]	DDR_SW_MAP	0	Sets the AHB bus master to DDR address space mapping mode 0–15.
[15:12]	DDR_HPD_MAP	0	Sets the HPDA master to DDR address space mapping mode 0–15.
[11:8]	DDR_DS_MAP	0	Sets the DSG master to DDR address space mapping mode 0–15.
7	DDRB_BUF_SZ	0x1	Configures the write buffer and read buffer size as per DDR burst size. This port is common for all buffers. IDC read buffer has fixed size of 32 bytes. Other buffers can be configured to 16-byte or 32-byte size. 0: Buffer size is configured to 16 bytes 1: Buffer size is configured to 32 bytes
6	DDRB_IDC_EN	0x1	Allows the read buffer for IDC interface in MSS DDR bridge to be disabled. Allowed values: 0: Disabled 1: Enabled
5	DDRB_SW_REN	0x1	Allows the read buffer for AHB BUS master in MSS DDR bridge to be disabled. Allowed values: 0: Disabled 1: Enabled
4	DDRB_SW_WEN	0x1	Allows the write combining buffer for AHB bus master in MSS DDR bridge to be disabled. Allowed values: 0: Disabled 1: Enabled
3	DDRB_HPD_REN	0x1	Allows the read buffer for high performance DMA master in MSS DDR bridge to be disabled. Allowed values: 0: Disabled 1: Enabled

Table 668 • DDRB_CR (continued)

2	DDRB_HPD_WEN	0x1	Allows the write combining buffer for high performance DMA master in MSS DDR bridge to be disabled. Allowed values: 0: Disabled 1: Enabled
1	DDRB_DS_REN	0x1	Allows the read buffer for DSG master in MSS DDR bridge to be disabled. Allowed values: 0: Disabled 1: Enabled
0	DDRB_DS_WEN	0x1	Allows write combining buffer for DSG master in MSS DDR bridge to be disabled. Allowed values: 0: Disabled 1: Enabled

22.3.16 EDAC Configuration Register

Table 669 • EDAC_CR

Bit Number	Name	Reset Value	Description
[31:7]	Reserved	0	
6	CAN_EDAC_EN	0	Allows the EDAC for CAN to be disabled. Allowed values: 0: Disabled 1: Enabled
5	USB_EDAC_EN	0	Allows the EDAC for USB to be disabled. Allowed values: 0: Disabled 1: Enabled
4	MAC_EDAC_RX_EN	0	Allows the EDAC for Ethernet Rx RAM to be disabled. Allowed values: 0: Disabled 1: Enabled
3	MAC_EDAC_TX_EN	0	Allows the EDAC for Ethernet Tx RAM to be disabled. Allowed values: 0: Disabled 1: Enabled
2	Reserved	0	
1	ESRAM1_EDAC_EN	0	Allows the EDAC for eSRAM1 to be disabled. Allowed values: 0: Disabled 1: Enabled
0	ESRAM0_EDAC_EN	0	Allows the EDAC for eSRAM0 to be disabled. Allowed values: 0: Disabled 1: Enabled

22.3.17 Master Weight Configuration Register 0

Table 670 • MASTER_WEIGHT0_CR

Bit Number	Name	Reset Value	Description
[31:30]	Reserved	0	
[29:25]	SW_WEIGHT_PDMA	0	Configures the round robin weight for peripheral DMA master. It is configurable from 1 to 32 (32 by default).

Table 670 • MASTER_WEIGHT0_CR (continued)

[24:20]	SW_WEIGHT_FAB_1	0	Configures the round robin weight for fabric (FIC_1) master. It is configurable from 1 to 32 (32 by default).
[19:15]	SW_WEIGHT_FAB_0	0	Configures the round robin weight for fabric (FIC_0) master. It is configurable from 1 to 32 (32 by default).
[14:10]	SW_WEIGHT_GIGE	0	Configures the round robin weight for Ethernet master. It is configurable from 1 to 32 (32 by default).
[9:5]	SW_WEIGHT_S	0	Configures the round robin weight for S master. It is configurable from 1 to 32 (32 by default).
[4:0]	SW_WEIGHT_IC	0	Configures the round robin weight for IC master. It is configurable from 1 to 32 (32 by default).

Note: The weight values are as given in [Table 672](#), page 692.

22.3.18 Master Weight Configuration Register 1

Table 671 • MASTER_WEIGHT1_CR

Bit Number	Name	Reset Value	Description
[31:15]	Reserved	0	
[14:10]	SW_WEIGHT_G	0	Configures the round robin weight for G master. It is configurable from 1 to 32 (32 by default).
[9:5]	SW_WEIGHT_USB	0	Configures the round robin weight for USB master. It is configurable from 1 to 32 (32 by default).
[4:0]	SW_WEIGHT_HPDM	0	Configures the round robin weight for HPDMA master. It is configurable from 1 to 32 (32 by default).

Note: The weight values are as given in [Table 672](#), page 692.

The following table provides weight values, where <master> is IC, S, GIGE, FIC_0, FIC_1, PDMA, HPDMA, USB, or G.

Table 672 • Programmable Weight Values

SW_WEIGHT_<master>	Weight
0	32
1	1
2	2
3	3
·	·
·	·
·	·
28	28
29	29
30	30
31	31

22.3.19 Software Interrupt Register

Table 673 • SOFT_IRQ_CR

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	SOFTINTERRUPT	0	1: FIIC SOFTINTERRUPT is asserted 0: SOFTINTERRUPT signal is cleared

22.3.20 Software Reset Control Register

Table 674 • SOFT_RESET_CR

Bit Number	Name	Reset Value	Description
[31:27]	Reserved	0	
26	MDDR_DDRFIC_SOFTRESET	0x1	0: Releases DDR_FIC controller from reset 1: Keeps DDR_FIC controller in reset
25	MDDR_CTLR_SOFTRESET	0x1	0: Releases MDDR controller from reset 1: Keeps MDDR controller in reset
24	MSS_GPOUT_31_24_SOFTRESET	0x1	0: Releases GPIO_OUT[31:24] from reset 1: Keeps GPIO_OUT[31:24] in reset
23	MSS_GPOUT_23_16_SOFTRESET	0x1	0: Releases GPIO_OUT[23:16] from reset 1: Keeps GPIO_OUT[23:16] in reset
22	MSS_GPOUT_15_8_SOFTRESET	0x1	0: Releases GPIO_OUT[15:8] from reset 1: Keeps GPIO_OUT[15:8] in reset
21	MSS_GPOUT_7_0_SOFTRESET	0x1	0: Releases GPIO_OUT[7:0] from reset 1: Keeps GPIO_OUT[7:0] in reset
20	MSS_GPIO_SOFTRESET	0x1	0: Releases the GPIO from reset, as long as it isn't being held in reset by some other means 1: Keeps the GPIO to be held in reset Asserting this soft reset bit holds APB register, GPIO input, and interrupt generation logic. This reset does not affect the GPIO OUT logic
19	FIC_1_SOFTRESET	0x1	0: Releases FIC_1 from reset 1: Keeps FIC_1 in reset
18	FIC_0_SOFTRESET	0x1	0: Releases FIC_0 from reset 1: Keeps FIC_0 in reset
17	HPDMA_SOFTRESET	0x1	0: Releases HPDMA from reset 1: Keeps HPDMA n reset
16	FPGA_SOFTRESET	0x1	0: Releases FPGA from reset 1: Keeps FPGA in reset
15	COMBLK_SOFTRESET	0	0: Releases COMM_BLK from reset 1: Keeps COMMUNICATION BLOCK (COMM_BLK) in reset
14	USB_SOFTRESET	0x1	0: Releases USB from reset 1: Keeps USB in reset

Table 674 • SOFT_RESET_CR (continued)

Bit Number	Name	Reset Value	Description
13	CAN_SOFTRESET	0x1	0: Releases CAN from reset 1: Keeps CAN in reset
12	I2C1_SOFTRESET	0x1	0: Releases I2C_1 from reset 1: Keeps I2C_1 in reset
11	I2C0_SOFTRESET	0x1	0: Releases I2C_0 from reset 1: Keeps I2C_0 in reset
10	SPI1_SOFTRESET	0x1	0: Releases SPI1 from reset 1: Keeps SPI1 in reset
9	SPI0_SOFTRESET	0x1	0: Releases SPI0 from reset 1: Keeps SPI0 in reset
8	MMUART1_SOFTRESET	0x1	0: Releases MMUART_1 from reset 1: Keeps MMUART_1 in reset
7	MMUART0_SOFTRESET	0x1	0: Releases MMUART_0 from reset 1: Keeps MMUART_0 in reset
6	TIMER_SOFTRESET	0x1	0: Releases the system timer from reset 1: Keeps the system timer in reset
5	PDMA_SOFTRESET	0x1	0: Releases the PDMA from reset 1: Keeps the PDMA in reset
4	MAC_SOFTRESET	0x1	0: Releases the Ethernet MAC from reset 1: Keeps the Ethernet MAC in reset
3	ESRAM1_SOFTRESET	0	0: Releases the eSRAM_1 memory controller from reset 1: Keeps the eSRAM_1 memory controller in reset
2	ESRAM0_SOFTRESET	0	0: Releases the eSRAM_0 memory controller from reset 1: Keeps the eSRAM_0 memory controller in reset
1	ENV1_SOFTRESET	0	0: Releases the eNVM_1 memory controller from reset 1: Keeps the eNVM_1 memory controller in reset
0	ENV0_SOFTRESET	0	0: Releases the eNVM_0 memory controller from reset 1: Keeps the eNVM_0 memory controller in reset

Reset values in the preceding table are the default values of the bits when peripherals are not configured using the software. If the peripheral is enabled using the software then the default reset value for that bit is 0x0.

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.21 M3 Configuration Register

Table 675 • M3_CR

Bit Number	Name	Reset Value	Description
[31:29]	Reserved	0	
28	M3_MPU_DISABLE	0	When set, disables the memory protection unit (MPU) within the Cortex-M3 processor.
[27:26]	STCLK_DIVISOR	0x3	Configures the amount of division to be performed on M3_CLK, in order to generate the STCLK input for the Cortex-M3 processor. This is used to control the frequency of STCLK. 00: M3_CLK/4 01: M3_CLK/8 10: M3_CLK/16 11: M3_CLK/32
[25:0]	STCALIB[25:0]	0x2000000	Used as the STCALIB input for the Cortex-M3 processor. It determines the rollover value for the internal SysTick timer of the Cortex-M3 processor. The bit definitions for this field are as follows: STCALIB[25] – NOREF bit of SysTick Calibration Value Register; 1 indicates STCLK is not provided. STCALIB[24] – SKEW bit of SysTick Calibration Value Register; 1 indicates calibration value is not exactly 10 ms. STCALIB[23:0] – TENMS field of SysTick Calibration Value Register; reload value to use for 10 ms timing.

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.22 Fabric Interface Control (FIC) Register

Table 676 • FAB_IF_CR

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	
[9:4]	SW_FIC_REG_SEL	0x38	Indicates whether a specific fabric region is accessible by FIC_0 or FIC_1. This register should not be changed during operation. 0: Fabric region associated with FIC_0 1: Fabric region associated with FIC_1 By default, fabric region 0, 1, 2 are accessible through FIC_0 and regions 3, 4, 5 are accessible through FIC_1. These bits are driven into the AHB bus in order to allocate a specific memory region to either FIC_0 or FIC_1.
3	FAB1_AHB_MODE	0	Controls whether the FIC_1 fabric interface supports AHB mode or APB mode. Allowed values: 0: Supports APB mode 1: Supports AHB mode
2	FAB0_AHB_MODE	0	Controls whether FIC_0 fabric interface supports AHB mode or APB mode. Allowed values: 0: Supports APB mode 1: Supports AHB mode

Table 676 • FAB_IF_CR (continued)

1	FAB1_AHB_BYPASS	0	0: FIC_1 is configured for synchronous bridging 1: FIC_1 is configured in bypass mode, if clock ratio is 1:1 and if in AHB mode
0	FAB0_AHB_BYPASS	0	0: FIC_0 is configured for synchronous bridging 1: FIC_0 is configured in bypass mode, if clock ratio is 1:1 and if in AHB mode

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.23 Loopback Control Register

Table 677 • LOOPBACK_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	
3	MSS_GPIOLOOPBACK	0	Controls whether internal loopback on the MSS GPIO is enabled. Allowed values: 0: No internal loopback 1: MSS GPIO outputs are looped back to MSS GPIO inputs
2	MSS_I2CLOOPBACK	0	Controls whether internal loopback between I2C_0 and I2C_1 is enabled. Allowed values: 0: No internal loopback 1: Traffic from I2C_0 is looped back to I2C_1 and vice versa
1	MSS_SPILOOPBACK	0	Controls whether internal loopback between SPI_0 and SPI_1 is enabled. Allowed values: 0: No loopback 1: SPI_0 Tx is looped back to SPI_1 Rx. SPI_1 Tx is looped back to SPI_0 Rx
0	MSS_MMUARTLOOPBACK	0	0: No loopback between MMUART_0 and MMUART_1 1: MMUART_0 Tx is looped back to MMUART_1 Rx MMUART_1 Tx is looped back to MMUART_0 Rx

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.24 GPIO System Reset Control Register

Table 678 • GPIO_SYSRESET_SEL_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	
3	MSS_GPIO_31_24_SYSRESET_SEL	0	0: Selects the combination of either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric to reset the GPIO 1: Causes GPIO[31:24] to be held in reset by the soft reset signal MSS_GPIO_31_24_SOFT_RESET

Table 678 • GPIO_SYSRESET_SEL_CR (continued)

2	MSS_GPIO_23_16_SYSRESET_SEL	0	0: Selects the combination of either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric to reset the GPIO 1: Causes GPIO[23:16] to be held in reset by the soft reset signal MSS_GPIO_23_16_SOFT_RESET
1	MSS_GPIO_15_8_SYSRESET_SEL	0	0: Selects the combination of either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric to reset the GPIO 1: Causes GPIO[15:8] to be held in reset by the soft reset signal MSS_GPIO_15_8_SOFT_RESET
0	MSS_GPIO_7_0_SYSRESET_SEL	0	0: Selects the combination of either power-on reset or the MSS_GPIO_RESET_N signal from the FPGA fabric to reset the GPIO 1: Causes GPIO[7:0] to be held in reset by the soft reset signal MSS_GPIO_7_0_SOFT_RESET

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.25 GPIO Input Source Select Control Register

Table 679 • GPIN_SRC_SEL_CR

Bit Number	Name	Reset Value	Description
[31:0]	MSS_GPINSOURCE	0	Used as select signal to generate a GPIO input signal by selecting two output signals from different IOMUXCELL or signals from I/O pads.

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.26 MDDR Configuration Register

Table 680 • MDDR_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	
3	PHY_SELF_REF_EN	0	Indicates that the DRAM has been put into self-refresh. This is used for automatic locking of the codes during intermediate runs for DDRC. Not used in non-DDRIO modes.
2	F_AXI_AHB_MODE	0	Used by the SMC_FIC, DDR_FIC, and DDR CTL to select the AXI/AHB interface in the fabric. Allowed values: 0: AHB interface is selected 1: AXI interface is selected
1	SDR_MODE	0	Used to select whether the MSS AXI interface accesses DDR memory or SDR memory (or other memory types) inside the fabric. Allowed values: 0: DDR memory is selected 1: SDR memory or other memory type is selected

Table 680 • MDDR_CR (continued)

0	MDDR_CONFIG_LOCAL	0x1	Configures whether the MSS AHBTOAPB2 bridge can directly access the APB slave within the MDDR subsystem or whether the APB slave is connected to the fabric. Allowed values: 0: AHBTOAPB2 bridge cannot access MDDR APB slave 1: AHBTOAPB2 bridge can access MDDR APB slave Reset signal for this bit is CC_RESET_N.
---	-------------------	-----	---

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.27 USB I/O Input Select Control Register

Table 681 • USB_IO_INPUT_SEL_CR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	
[1:0]	USB_IO_INPUT_SEL	0	Selects one of the four USB data interfaces from IOMUXCELLs and I/O pads. Depending on the device and package, not all interfaces may be available. Allowed values: 00: USBA interface can be connected to USB 01: USBB interface can be connected to USB 10: USBC interface can be connected to USB 11: USBD interface can be connected to USB

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.28 Peripheral Clock MUX Select Control Register

Table 682 • PERIPH_CLK_MUX_SEL_CR

Bit Number	Name	Reset Value	Description
[31:3]	Reserved	0	
2	TRACECLK_DIV2_SEL	0	Selects whether the Cortex-M3 processor trace clock source is M3_CLK or M3_CLK/2. Allowed values: 0: M3_CLK selected as source of TRACECLKIN_I 1: M3_CLK/2 selected as source of TRACECLKIN_I
1	SPI1_SCK_FAB_SEL	0	Selects the SPI1_SCK from the fabric or I/O pads. Allowed values: 0: SPI1_SCK clock from I/O pads is selected and fed to SPI1 1: SPI1_SCK clock from the fabric is selected and fed to SPI1
0	SPI0_SCK_FAB_SEL	0	Selects the SPI0_SCK from the fabric or I/O pads. Allowed values: 0: SPI0_SCK clock from I/O pads is selected and fed to SPI0 1: SPI0_SCK clock from fabric is selected and fed to SPI0

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.29 Watchdog Configuration Register

Table 683 • WDOG_CR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	
1	WDOGMODE	0	Resets/interrupts the mode selection bit from System Register. This value can be read from the WDOGCONTROL register within the WatchDog module.
0	WDOGENABLE	0	Enables the bit for Watchdog module. The status of this bit can be monitored in the WDOGENABLE register within the WatchDog module.

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

Information about other WatchDog system registers is given in [Table 745](#), page 727 and [Table 746](#), page 727.

22.3.30 MDDR I/O Calibration Control Register

Table 684 • MDDR_IO_CALIB_CR

Bit Number	Name	Reset Value	Description
[31:15]	Reserved	0	
14	CALIB_LOCK	0	Used in the DDRIO calibration block as an override to lock the codes during intermediate runs. When the firmware receives CALIB_INTRPT, it may choose to assert this signal by prior knowledge of the traffic without going through the process of putting the DDR into self refresh. This bit is only read/write.
13	CALIB_START	0	Used in the DDRIO calibration block and indicates that rerun of the calibration state machine is required.
12	CALIB_TRIM		Used in the DDRIO calibration block and indicates the override of the calibration value from the PC code/programmed code values.
[11:6]	NCODE	0	Used in the DDRIO calibration block and indicates DPC override NCODE from flash. This can also be overwritten from the firmware.
[5:0]	PCODE	0	Used in the DDRIO calibration block and indicates PC override PODE from flash. This can also be overwritten from the firmware.

22.3.31 EDAC Interrupt Enable Control Register

Table 685 • EDAC_IRQ_ENABLE_CR

Bit Number	Name	Reset Value	Description
[31:15]	Reserved	0	

Table 685 • EDAC_IRQ_ENABLE_CR (continued)

Bit Number	Name	Reset Value	Description
14	MDDR_ECC_INT_EN	0	Allows the error EDAC for MDDR status update to be disabled. Allowed values: 0: Disabled 1: Enabled
13	CAN_EDAC_2E_EN	0	Allows the 2-bit error EDAC for CAN status update to be disabled. Allowed values: 0: Disabled 1: Enabled
12	CAN_EDAC_1E_EN	0	Allows the 1-bit error EDAC for CAN status update to be disabled. Allowed values: 0: Disabled 1: Enabled
11	USB_EDAC_2E_EN	0	Allows the 2-bit error EDAC for USB status update to be disabled. Allowed values: 0: Disabled 1: Enabled
10	USB_EDAC_1E_EN	0	Allows the 1-bit error EDAC for USB status update to be disabled. Allowed values: 0: Disabled 1: Enabled
9	MAC_EDAC_RX_2E_EN	0	Allows the 2-bit error EDAC for Ethernet Rx RAM status update to be disabled. Allowed values: 0: Disabled 1: Enabled
8	MAC_EDAC_RX_1E_EN	0	Allows the 1-bit error EDAC for Ethernet Rx RAM status update to be disabled. Allowed values: 0: Disabled 1: Enabled
7	MAC_EDAC_TX_2E_EN	0	Allows the 2-bit error EDAC for Ethernet Tx RAM status update to be disabled. Allowed values: 0: Disabled 1: Enabled
6	MAC_EDAC_TX_1E_EN	0	Allows the 1-bit error EDAC for Ethernet Tx RAM status update to be disabled. Allowed values: 0: Disabled 1: Enabled
5	Reserved	0	
4	Reserved	0	
3	ESRAM1_EDAC_2E_EN	0	Allows the 2-bit error EDAC for eSRAM1 status update to be disabled. Allowed values: 0: Disabled 1: Enabled
2	ESRAM1_EDAC_1E_EN	0	Allows the 1-bit error EDAC for eSRAM1 status update to be disabled. Allowed values: 0: Disabled 1: Enabled

Table 685 • EDAC_IRQ_ENABLE_CR (continued)

Bit Number	Name	Reset Value	Description
1	ESRAM0_EDAC_2E_EN	0	Allows the 2-bit error EDAC for eSRAM0 status update to be disabled. Allowed values: 0: Disabled 1: Enabled
0	ESRAM0_EDAC_1E_EN	0	Allows the 1-bit error EDAC for eSRAM0 status update to be disabled. Allowed values: 0: Disabled 1: Enabled

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.32 USB Configuration Register

Table 686 • USB_CR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	
1	USB_DDR_SELECT	0	Used to configure USB works in single data rate (SDR) mode or double data rate (DDR) mode. Allowed values: 0: SDR mode is selected 1: DDR mode is selected
0	USB_UTMI_SEL	0	Used to configure the USB interface as ULPI PHY or UTMI interface. Allowed values: 0: ULPI PHY interface is selected 1: UTMI interface is selected

22.3.33 eSRAM PIPELINE Configuration Register

Table 687 • ESRAM_PIPELINE_CR

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	ESRAM_PIPELINE_ENABLE	0x1	Controls the pipeline in the read path of eSRAM memory. Allowed values: 0: Pipeline is bypassed 1: Pipeline is present in the memory read path

MSS Interrupt Enable Control Register

Table 688 • MSS_IRQ_ENABLE_CR

Bit Number	Name	Reset Value	Description
[31:20]	Reserved	0	
[19:10]	DDRB_INTERRUPT_EN	0x3FF	Used to mask the MSS DDR bridge interrupt to the Cortex-M3 processor
[9:7]	CC_INTERRUPT_EN	0x7	Used to mask the cache interrupt to the Cortex-M3 processor
[6:0]	SW_INTERRUPT_EN	0x7F	Used to mask the AHB bus interrupt to the Cortex-M3 processor

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.34 RTC Wake Up Configuration Register

Table 689 • RTC_WAKEUP_CR

Bit Number	Name	Reset Value	Description
[31:3]	Reserved	0	
2	RTC_WAKEUP_C_EN	0	Enables RTC_WAKEUP interrupt to the system controller
1	RTC_WAKEUP_FAB_EN	0	Enables the RTC_WAKEUP interrupt to the fabric
0	RTC_WAKEUP_M3_EN	0	Enables the RTC_WAKEUP interrupt to the Cortex-M3 processor

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.35 MAC Configuration Register

Table 690 • MAC_CR

Bit Number	Name	Reset Value	Description
[31:9]	Reserved	0	
[8:5]	RGMII_TXC_DELAY_SEL	0	Specifies how many delay taps the RGMII transmit clock passes through 0 to 15
[3:2]	ETH_PHY_MODE	0	Indicates the Ethernet PHY mode. Allowed values: 000: RMII 001: Reserved 010: TBI 011: MII 100: GMII Other values: Reserved
[1:0]	ETH_LINE_SPEED	0	Indicates the Ethernet line speed. Allowed values: 00: 10 Mbps 01: 100 Mbps 10: 1,000 Mbps 11: Reserved

22.3.36 MSS DDR PLL Status Low Configuration Register

This register is to be configured by flash bits only and you should not write to it while the source clock is active.

Table 691 • MSSDDR_PLL_STATUS_LOW_CR

Bit Number	Name	Reset Value	Description
[31:30]	Reserved	0	
[29:26]	FACC_PLL_LOCKCNT	0	Configures the MPLL LOCK counter value given by ($2^{\text{binary value}} + 5$). For example, if the binary value is 0000, the LOCK counter value is 32, and if binary value is 1111, then its value is 1,048,576.
[25:23]	FACC_PLL_LOCKWIN	0	Configures the MPLL phase error window for LOCK assertion as a fraction of the divided reference period. Values are at typical PVT only and are not PVT compensated. 000: 500 ppm 100: 8000 ppm 001: 1000 ppm 101: 16000 ppm 010: 2000 ppm 110: 32000 ppm 011: 4000 ppm 111: 64000 ppm
[22:19]	FACC_PLL_RANGE	0	Configures the MPLL filter range. The bit definitions are in Table 692 , page 704.
[19:16]	FACC_PLL_DIVQ	0x2	Configures the MPLL output divider value in order to generate the DDR clock. Output divider values are given by: 000: Divided by 1 001: Divided by 2 010: Divided by 4 011: Divided by 8 100: Divided by 16 101: Divided by 32 While it is possible to configure the MPLL output divider as $\div 1$, this setting must not be used when the DDR is operational. This is to ensure that the clock to the DDR has an even mark:space ratio.
[15:6]	FACC_PLL_DIVF	0x2	Configures the MPLL feedback divider value, which is given by the binary value +1. The binary value ranges from 0000000000, which is the divisor value of 1, to 1111111111, which is the divisor value of 1,024.
[5:0]	FACC_PLL_DIVR	0x1	Configures the MPLL reference divider value, which is given by binary value +1. For example, if the value is 00000, then the divisor value is 1 (00000 + 1). Both REFCLK and post-divide REFCLK must be within the range specified in the SmartFusion2 datasheet.

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.36.1 FACC_PLL_RANGE

Table 692 • FACC_PLL_RANGE

Bits[23:19]	PLL Range
0000	Bypass
0111	18–29 MHz
0001	1–1.6 MHz
1000	29–46 MHz
0010	1.6–2.6 MHz
1001	46–75 MHz
0011	2.6–4.2 MHz
1010	75–120 MHz
0100	4.2–6.8 MHz
1011	120–200 MHz
0101	6.8–11 MHz
0110	11–18 MHz

22.3.37 MSS DDR PLL Status High Configuration Register

Table 693 • MSSDDR_PLL_STATUS_HIGH_CR

Bit Number	Name	Reset Value	Description
[31:13]	Reserved	0	
[12:8]	FACC_PLL_SSMF	0	Drives the spread spectrum modulation frequency (SSMF) input of the MPLL. The only allowable value to be programmed in this field is 0, as spread spectrum mode is not supported for the MPLL.
[7:6]	FACC_PLL_SSMD	0	Drives the spread spectrum modulation depth (SSMD) input of the MPLL. The only allowable value to be programmed in this field is 0, as spread spectrum mode is not supported for the MPLL.
5	FACC_PLL_SSE	0	Drives the SSE input of the MPLL. The only allowable value to be programmed in this field is 0, as spread spectrum mode is not supported for the MPLL.
4	FACC_PLL_PD	0	A PD signal is provided for lowest quiescent current. When PD is asserted, the MPLL powers down and outputs will be Low. PD has precedence over all other functions.
3	FACC_PLL_FSE	0	Configures PLL internal and external feedback paths. The only allowed value to be programmed in this field is 1.
2	FACC_PLL_MODE_3V3	0x1	Configures MPLL analog operational voltage. 1: 3.3 V 0: 2.5 V
1	FACC_PLL_MODE_1V2	0x1	Configures the PLL core voltage. 1: 1.2 V Do not write to this field.

Table 693 • MSSDDR_PLL_STATUS_HIGH_CR (continued)

Bit Number	Name	Reset Value	Description
0	FACC_PLL_BYPASS	0	Powers down the MPLL core and bypasses it such that PLLOUT tracks REFCLK.

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.38 MSS DDR Fabric Alignment Clock Controller (FACC) Configuration Register 1

Table 694 • MSSDDR_FACC1_CR

Bit Number	Name	Reset Value	Description
[31:28]	Reserved	0	
27	FACC_FAB_REF_SEL	0	Selects the source of the reference clock to be supplied to the MPLL. Allowed values are: 0: 50 MHz RC 1: Fabric clock (CLK_BASE)
26	CONTROLLER_PLL_INIT	0x1	Indicates whether the FACC is to be configured for PLL initialization mode. The user can write to it when it detects that the MPLL has lost lock and it wants to switch to a known good clock source until the MPLL comes back into lock. This causes the 50 MHz clock to be selected through to the MSS. It also interrupts the System Controller, which then waits for the MPLL to come into lock before clearing this bit and thereby selecting the MPLL output as the MSS clock source again. The allowed values of this bit are: 0: The corresponding FACC multiplexer select lines or clock gate control line comes from the normal run-time configuration signals (from relevant MSS system register bits). 1: The corresponding FACC multiplexer select lines or clock gate control line are overridden by hardwired PLL initialization selection, as described below: – Override the four no-glitch multiplexers related to the aligned clocks, so that they select CLK_STANDBY as the source of M3_CLK, APB_0_CLK, APB_0_CLK and DDR_SMC_FIC_CLK. – Override the selection of the FACC standby multiplexer, so that it selects the RCOSC_25_50MHZ clock as the source of CLK_STANDBY. – Override the selection of the FACC reference multiplexer, so that it selects CLK_BASE clock as the source of MPLL_REF_CLK. – Override the value of the PLL bypass configuration signal, so that it forces the MPLL bypass path not to be used. – Force MDDR_CLK to be gated off.

Table 694 • MSSDDR_FACC1_CR (continued)

Bit Number	Name	Reset Value	Description
25	PERSIST_CC	0	Feeds into the MSS Reset Controller. Based on the value of PERSIST_CC, the Reset Controller asserts a reset (CC_RESET_N) to the FACC (which inverts it and passes it on to the PLL as MSSDDR_PLL_RESET), either on every MSS system reset or just on power-up reset. This field is to be configured using flash bits. Do not write to this field. The only allowable value for this bit is 1. The reset signal for this register is PORESET_N.
[24:22]	BASE_DIVISOR	0	Indicates the ratio between CLK_A and the re-generated version of CLK_BASE, called CLK_BASE_REGEN. Do not write to this field. The allowed values are listed in Table 695 , page 707.
[21:19]	DDR_FIC_DIVISOR	0	Indicates the ratio between CLK_A and DDR_SMC_FIC_CLK. The user can write to this field dynamically during run time, even when the source clock is active. The allowed values are listed in Table 695 , page 707.
[18:16]	FIC_1_DIVISOR	0	Indicates the ratio between CLK_A and the clock being used in the fabric, to clock the soft IP block which is interfacing to FIC_1 of the MSS. The user can write to this field dynamically during run time, even when the source clock is active. The allowed ratios for CLK_A: fabric clock (FIC_1) is listed in Table 695 , page 707.
[15:13]	FIC_0_DIVISOR	0	Indicates the ratio between CLK_A and the clock being used in the fabric, to clock the soft IP block which is interfacing to FIC_0 of the MSS. The user can write to this field dynamically during run time, even when the source clock is active. The allowed ratios for CLK_A: fabric clock (FIC_0) are listed in Table 695 , page 707.
12	FACC_GLMUX_SEL	0	Contains the select line for the four no-glitch multiplexers within the FACC, which are related to the aligned clocks. All four of these multiplexers are switched by one signal. Allowed values: 1: M3_CLK, APB_0_CLK, APB_1_CLK, DDR_SMC_FIC_CLK all driven from CLK_STANDBY 0: M3_CLK, APB_0_CLK, APB_1_CLK, DDR_SMC_FIC_CLK all driven from stage B dividers Configure this field using flash bits. Do not write to this field.
[11:9]	M3_CLK_DIVISOR	0	Indicates the ratio between CLK_A and M3_CLK. The user can write to this field dynamically during run time, even when the source clock is active.
8	DDR_CLK_EN	0	Determines whether or not the clock to the MDDR block is to be gated off. Allowed values: 0: MDDR_CLK is gated off 1: MDDR_CLK is allowed to propagate through to MDDR block Do not write to this field dynamically while the source clock is active.
[7:5]	APB1_DIVISOR	0	Indicates the ratio between CLK_A and APB_1_CLK. The user can write to this field dynamically during run time, even when the source clock is active. The allowed values are described in Table 695 , page 707.

Table 694 • MSSDDR_FACC1_CR (continued)

Bit Number	Name	Reset Value	Description
[4:2]	APB0_DIVISOR	0	Indicates the ratio between CLK_A and APB_0_CLK. The user can write to this field dynamically during run time, even when the source clock is active. The allowed values are described in Table 695 , page 707.
[1:0]	DIVISOR_A	0	Indicates the ratio between CLK_SRC and CLK_A. Allowed values: 00: 1:1 01: 2:1 10: 3:1 11: Reserved Configure this field statically. Do not write to this field while the source clock is active.

Note: Do not change these register fields dynamically for 005 and 010 devices, see [System Registers Behavior for M2S005/010 Devices](#), page 682.

22.3.38.1 Clock Ratio

Table 695 • Clock Ratio

Bits	Clock Ratio
000	1:1
001	2:1
010	4:1
100	8:1
101	16:1
110	32:1
Other values	Reserved

22.3.39 MSS DDR Fabric Alignment Clock Controller Configuration Register 2

Table 696 • MSSDDR_FACC2_CR

Bit Number	Name	Reset Value	Description
[31:14]	Reserved	0	
13	MSS_XTAL_RTC_EN	0x1	Enable signal for auxiliary crystal oscillator (RTC crystal oscillator)
12	MSS_XTAL_EN	0x1	Enables the signal for the main crystal oscillator. If the main crystal oscillator is selected as the MSS Flash*Freeze clock source, this bit must be asserted at all times (even when not in Flash*Freeze mode). 1: Enable 0: Disable
11	MSS_CLK_ENVM_EN	0x1	Enables internal eNVM RC oscillator. Configure this field statically. Do not write to this field while the source clock is active.

Table 696 • MSSDDR_FACC2_CR (continued)

Bit Number	Name	Reset Value	Description
10	MSS_1MHZ_EN	0x1	Enables the signal for the 1 MHz RC oscillator. If the 1 MHz RC oscillator is selected as the MSS Flash*Freeze clock source, this bit must be asserted at all times (even when not in Flash*Freeze mode). 1: Enable 0: Disable
9	MSS_25_50MHZ_EN	0x1	Enables the signal for the 50 MHz RC oscillator. If the 50 MHz RC oscillator is selected as the MSS Flash*Freeze clock source, this bit must be asserted at all times (even when not in Flash*Freeze mode). 1: Enable 0: Disable
[8:6]	FACC_STANDBY_SEL	0	Contains the select lines for the three 2 to 1 no-glitch multiplexers, which implement the 4 to 1 no-glitch standby MUX function. This is used to allow one of 4 possible clocks to proceed through to the MSS during FACC PLL Initialization Time. There are two MUXes in the first rank and these feed into a third MUX in the second rank. Bit 6 feeds into one of the first rank 2 to 1 MUXes (Standby MUX 0) and is defined as follows: 0: MUX 0 output comes from RCOSC_25_50MHZ 1: MUX 0 output comes from XTLOSC_CLK Bit 7 feeds into one of the first rank 2 to 1 MUXes (Standby MUX 1) and is defined as follows: 0: MUX 1 output comes from RCOSC_1MHZ Bit 8 feeds into the second rank 2 to 1 MUX (Standby MUX 2) and is defined as follows: 0: MUX 2 output comes from MUX 0 1: MUX 2 output comes from MUX 1 Do not write to this field while the standby clock is active.
5	FACC_PRE_SRC_SEL	0	Must always be 0. Allowed values: 0: RCOSC_1MHZ is fed through to the source no-glitch clock multiplexer.
[4:2]	FACC_SRC_SEL	0	Contains the select lines for the three 2 to 1 no-glitch multiplexers, which implement a 4 to 1 no-glitch source MUX function. There are two MUXes in the first rank and these feed into a third MUX in the second rank. Bit 2 feeds into one of the first rank 2 to 1 MUXes (Source MUX 0) and is defined as follows: 0: MUX 0 output comes from RCOSC_25_50MHZ 1: MUX 0 output comes from XTLOSC_CLK Bit 3 feeds into one of the first rank 2 to 1 MUXes (Source MUX 1) and is defined as follows: 0: MUX 1 output comes from RCOSC_1MHZ 1: MUX 1 output comes from MPLL_OUT_CLK Bit 4 feeds into the second rank 2 to 1 MUX (Source MUX 2) and is defined as follows: 0: MUX 2 output comes from MUX 0 1: MUX 2 output comes from MUX 1 When switching any of the no-glitch MUXes, both the clock being switched from and the clock being switched to must be running. Do not write to this field while the source clock is active.

Table 696 • MSSDDR_FACC2_CR (continued)

Bit Number	Name	Reset Value	Description
[1:0]	RTC_CLK_SEL	0	Indicates which of the possible clocks are to be configured as the source of the MSS RTC clock. The allowed values are as follows: 00: RTC_CLK comes from XTLOSC_CLK 01: RTC_CLK comes from RCOSC_1MHZ. 10: RTC_CLK comes from RCOSC_25_50MHZ 11: RTC_CLK comes from RTC_XTLOSC_CLK The reset signal for this bit is SYSRESET_N.

22.3.40 PLL LOCK Enable Control Register

Table 697 • PLL_LOCK_EN_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	
3	FAB_PLL_LOCK_LOST_EN	0	Masking signal to enable Fabric PLL LOCK LOST interrupt to Cortex-M3 processor.
2	FAB_PLL_LOCK_EN	0	Masking signal to enable Fabric PLL LOCK interrupt to Cortex-M3 processor.
1	MPLL_LOCK_LOST_EN	0	Masking signal to enable MPLL LOCK LOST interrupt to Cortex-M3 processor.
0	MPLL_LOCK_EN	0	Masking signal to enable MPLL LOCK interrupt to Cortex-M3 processor.

22.3.41 MSS DDR Clock Calibration Control Register

Table 698 • MSSDDR_CLK_CALIB_CR

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	FAB_CALIB_START	0	Writing to this bit causes a one clock tick pulse to be generated on FABCALIBSTART. This is used to start an FPGA fabric calibration test circuit.

22.3.42 PLL Delay Line Select Control Register

Table 699 • PLL_DELAY_LINE_SEL_CR

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	
3:2	PLL_FB_DEL_SEL	0	Must be programmed to a specific value by Libero SoC and never be modified after that.
1:0	PLL_REF_DEL_SEL	0	Must be programmed to a specific value by Libero SoC and never be modified after that.

22.3.43 MAC Status Clear on Read Control Register

Table 700 • MAC_STAT_CLRONRD_CR

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	MAC_STAT_CLRONRD	0x1	MAC statistics counters that have been set are cleared after they are read.

22.3.44 Reset Source Control Register

Table 701 • RESET_SOURCE_CR

Bit Number	Name	Reset Value	Description
[31:8]	Reserved	0	
7	USER_M3_RESET_DETECT	0x1	Indicates that an M3 user reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. Reset signal: FAB_M3_RESET_M3_CLK_N.
6	USER_RESET_DETECT	0x1	Indicates that a MSS user reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. Reset signal: USER_MSS_RESET_M3_CLK_N.
5	WDOG_RESET_DETECT	0x1	Indicates that a Watchdog reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. Reset signal: WDOGTIMEOUT_M3_CLK_N.
4	LOCKUP_RESET_DETECT	0x1	Indicates that a Cortex-M3 processor lockup reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. The reset signal for this bit is LOCKUP_DEL2_N.
3	SOFT_RESET_DETECT	0x1	Indicates that a soft reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. The reset signal for this bit is SYSRESETREQ_DEL2_N.
2	CONTROLLER_M3_RESET_DETECT	0x1	Indicates that a controller M3 reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. The reset signal for this bit is M3_RESET_M3_CLK_N.
1	CONTROLLER_RESET_DETECT	0x1	Indicates that an MSS controller reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. Reset signal: SC_MSS_RESET_M3_CLK_N.
0	PO_RESET_DETECT	0x1	Indicates that a power-up reset has occurred. During the device boot sequence, this register should be cleared to arm it to detect the next reset event. The reset signal for this bit is PO_RESET_M3_CLK_N.

22.3.45 Dcode Bus Error Address Status Register

Table 702 • CC_DC_ERR_ADDR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_DC_ERR_ADDR	0	Stores the address from the DCode bus on which an error has occurred.

22.3.46 ICode Bus Error Address Status Register

Table 703 • CC_IC_ERR_ADDR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_IC_ERR_ADDR	0	Stores the address from the ICode bus on which an error has occurred.

22.3.47 System Bus Error Address Status Register

Table 704 • CC_SB_ERR_ADDR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_SB_ERR_ADDR	0	Stores the address from the system bus on which an error has occurred.

22.3.48 ICode Miss Control Status Register

Table 705 • CC_IC_MISS_CNTR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_IC_MISS_CNT	0	Counts the total number of cache misses that occurs on the cacheable region through the ICode bus. Rolls back after maximum value. This counter is put to reset value by setting the CC_IC_MISS_CNTCLR bit.

22.3.49 ICode Hit Control Status Register

Table 706 • CC_IC_HIT_CNTR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_IC_HIT_CNT	0	Keeps count of the total number of cache hits that occurs on the cacheable region through the ICode bus. Rolls back after maximum value. This counter is put to the reset value by setting CC_IC_HIT_CNTCLR.

22.3.50 DCode Miss Control Status Register

Table 707 • CC_DC_MISS_CNTR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_DC_MISS_CNT	0	Counts the total number of cache misses that occurs on the cacheable region through the DCode bus. Rolls back after maximum value. This counter is put to the reset value by setting CC_DC_MISS_CNTCLR.

22.3.51 DCode Hit Control Status Register

Table 708 • CC_DC_HIT_CNTR_CR

Bit Number	Name	Reset Value	Description
[31:0]	CC_DC_HIT_CNT	0	Counts the total number of cache hits that occurs on the cacheable region through the DCode bus. Rolls back after maximum value. This counter is put to reset value by setting CC_DC_HIT_CNTCLR.

22.3.52 ICode Transaction count Control Status Register

Table 709 • CC_IC_TRANS_CNTR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_IC_TRANS_CNT	0	Keeps count of the total number of transaction counts processed by the cache engine (cacheable and non-cacheable reads on ICode bus). This counter is put to the reset value by setting CC_IC_TRAN_CNTCLR.

22.3.53 DCode Transaction Count Control Status Register

Table 710 • CC_DC_TRANS_CNTR_SR

Bit Number	Name	Reset Value	Description
[31:0]	CC_DC_TRANS_CNT	0	Keeps count of the total number of transaction counts processed by the cache engine (cacheable and non-cacheable reads on DCode bus). This counter is put to the reset value by setting CC_DC_TRANS_CNTCLR bit.

22.3.54 MSS DDR Bridge DS master Error Address Status Register

Table 711 • DDRB_DS_ERR_ADR_SR

Bit Number	Name	Reset Value	Description
[31:0]	DDRB_DS_ERR_ADD	0	If a write transfer initiated at the MSS DDR bridge arbiter interface to empty data present in the write buffer of the DS master which receives an error response, the address for which error response is received is placed in this register. Address indicates TAG value for which error response is received. The following values are updated in this register as per buffer size: 16 bytes: DDRB_DS_ERR_ADR[31:4] = TAG, DDRB_DS_ERR_ADR [3:0] = 0000 32 bytes: DDRB_DS_ERR_ADR [31:5] = TAG[27:1], DDRB_DS_ERR_ADR[4:0] = 0000.

22.3.55 MSS DDR Bridge High Performance DMA Master Error Address Status Register

Table 712 • DDRB_HPD_ERR_ADR_SR

Bit Number	Name	Reset Value	Description
[31:0]	DDRB_HPD_ERR_ADD	0	If a write transfer initiated at the MSS DDR bridge arbiter interface to empty data present in the write buffer of the HPDMA master which receives an error response, the address for which the error response is received is placed in this register. Address indicates TAG value for which error response is received. The following values are updated in this register as per buffer size: 16 bytes: DDRB_HPD_ERR_ADR[31:4] = TAG, DDRB_HPD_ERR_ADR[3:0] = 0000 32 bytes: DDRB_HPD_ERR_ADR[31:5] = TAG[27:1], DDRB_HPD_ERR_ADR[4:0] = 0000.

22.3.56 MSS DDR Bridge AHB Bus Error Address Status Register

Table 713 • DDRB_SW_ERR_ADR_SR

Bit Number	Name	Reset Value	Description
[31:0]	DDRB_SW_ERR_ADD	0	If a write transfer initiated at the MSS DDR bridge arbiter interface to empty data present in the write buffer allocated for the AHB bus, which receives an error response, the address for which the error response is received is placed in this register. Address indicates TAG value for which the error response is received. The following values are updated in this register as per buffer size: 16 bytes: DDRB_SW_ERR_ADR[31:4] = TAG, DDRB_SW_ERR_ADR [3:0] = 0000 32 bytes: DDRB_SW_ERR_ADR [31:5] = TAG[27:1], DDRB_SW_ERR_ADR [4:0] = 0000.

22.3.57 MSS DDR Bridge Buffer Empty Status Register

Table 714 • DDRB_BUF_EMPTY_SR

Bit Number	Name	Reset Value	Description
[31:7]	Reserved	0	
6	DDRB_IDC_RBEMPTY	0	When set to '1', indicates that the read buffer of the IDC master does not have valid data.
5	DDRB_HPD_RBEMPTY	0	When set to '1', indicates that the read buffer of the HPDMA master does not have valid data.
4	DDRB_HPD_WBEMPTY	0	When set to '1', indicates that the write buffer of the HPDMA master does not have valid data.
3	DDRB_SW_RBEMPTY	0	When set to '1', indicates that the read buffer of the AHB bus matrix master does not have valid data.
2	DDRB_SW_WBEMPTY	0	When set to '1', indicates that the write buffer of the AHB bus matrix master does not have valid data.
1	DDRB_DS_RBEMPTY	0	When set to '1', indicates that the read buffer of the DSG master does not have valid data.
0	DDRB_DS_WBEMPTY	0	When set to '1', indicates that the write buffer of the DSG master does not have valid data.

22.3.58 MSS DDR Bridge Disable Buffer Status Register

Table 715 • DDRB_DSBL_DN_SR

Bit Number	Name	Reset Value	Description
[31:7]	Reserved	0	
6	DDRB_IDC_DSBL_DN	0	Is set to '1' once the AHB bus matrix read buffer is disabled after getting a read buffer disable command from processor.
5	DDRB_HPD_RDSBL_DN	0	Is set to '1' once the HPDMA read buffer is disabled after getting a read buffer disable command from processor.
4	DDRB_HPD_WDSBL_DN	0	Is set to '1' once the HPDMA write buffer is disabled after getting a write buffer disable command from processor.
3	DDRB_SW_RDSBL_DN	0	Is set to '1' once the AHB bus matrix read buffer is disabled after getting a read buffer disable command from processor.
2	DDRB_SW_WDSBL_DN	0	Is set to '1' once the AHB bus matrix write buffer is disabled after getting a write buffer disable command from processor.
1	DDRB_DS_RDSBL_DN	0	Is set to '1' once the DS read buffer is disabled after getting a read buffer disable command from processor.
0	DDRB_DS_WDSBL_DN	0	Is set to '1' once the DS write buffer is disabled after getting a write buffer disable command from processor.

22.3.59 eSRAM0 EDAC Count

Table 716 • ESRAM0_EDAC_CNT

Bit Number	Name	Reset Value	Description
[31:16]	ESRAM0_EDAC_CNT_2E	0	16-bit counter value in eSRAM0 incremented by eSRAM0 EDAC 2-bit error. The counter will not roll back and will stay at its maximum value.
[15:0]	ESRAM0_EDAC_CNT_1E	0	16-bit counter value in eSRAM0 incremented by eSRAM0 EDAC 1-bit error. The counter will not roll back and will stay at its maximum value.

22.3.60 eSRAM1 EDAC Count

Table 717 • ESRAM1_EDAC_CNT

Bit Number	Name	Reset Value	Description
[31:16]	ESRAM1_EDAC_CNT_2E	0	16-bit counter value in eSRAM1 incremented by eSRAM1 EDAC 2-bit error. The counter will not roll back and will stay at its maximum value.
[15:0]	ESRAM1_EDAC_CNT_1E	0	16-bit counter value in eSRAM1 incremented by eSRAM1 EDAC 1-bit error. The counter will not roll back and will stay at its maximum value.

22.3.61 MAC EDAC Transmitter Count

Table 718 • MAC_EDAC_TX_CNT

Bit Number	Name	Reset Value	Description
[31:16]	MAC_EDAC_TX_CNT_2E	0	16-bit counter that counts the number of 2-bit errors for Ethernet (MAC TX EDAC). The counter will not roll back and will stay at its maximum value.
[15:0]	MAC_EDAC_TX_CNT_1E	0	16-bit counter that counts the number of 1-bit errors for Ethernet (MAC TX EDAC). The counter will not roll back and will stay at its maximum value.

22.3.62 MAC EDAC Receiver Count

Table 719 • MAC_EDAC_RX_CNT

Bit Number	Name	Reset Value	Description
[31:16]	MAC_EDAC_RX_CNT_2E	0	16-bit counter that counts the number of 2-bit errors for Ethernet (MAC RX EDAC). The counter will not roll back and will stay at its maximum value.
[15:0]	MAC_EDAC_RX_CNT_1E	0	16-bit counter that counts the number of 1-bit errors for Ethernet (MAC RX EDAC). The counter will not roll back and will stay at its maximum value.

22.3.63 USB EDAC Count

Table 720 • USB_EDAC_CNT

Bit Number	Name	Reset Value	Description
[31:16]	USB_EDAC_CNT_2E	0	16-bit counter that counts the number of 2-bit errors for USB. The counter will not roll back and will stay at its maximum value.
[15:0]	USB_EDAC_CNT_1E	0	16-bit counter that counts the number of 1-bit errors for USB. The counter will not roll back and will stay at its maximum value.

22.3.64 CAN EDAC Count

Table 721 • CAN_EDAC_CNT

Bit Number	Name	Reset Value	Description
[31:16]	CAN_EDAC_CNT_2E	0	16-bit counter that counts the number of 2-bit errors for CAN. The counter will not roll back and will stay at its maximum value.
[15:0]	CAN_EDAC_CNT_1E	0	16-bit counter that counts the number of 1-bit errors for CAN. The counter will not roll back and will stay at its maximum value.

22.3.65 eSRAM0 EDAC Address Register

Table 722 • ESRAM0_EDAC_ADR

Bit Number	Name	Reset Value	Description
[31:26]	Reserved	0	
[25:13]	ESRAM0_EDAC_2E_AD	0	Stores the address from eSRAM0 on which a 2-bit SECDED error has occurred.
[12:0]	ESRAM0_EDAC_1E_AD	0	Stores the address from eSRAM0 on which a 1-bit SECDED error has occurred.

22.3.66 eSRAM1 EDAC Address Register

Table 723 • ESRAM1_EDAC_ADR

Bit Number	Name	Reset Value	Description
[31:26]	Reserved	0	
[25:13]	ESRAM1_EDAC_2E_AD	0	Stores the address from eSRAM1 on which a 2-bit SECDED error has occurred.
[12:0]	ESRAM1_EDAC_1E_AD	0	Stores the address from eSRAM1 on which a 1-bit SECDED error has occurred.

22.3.67 MAC EDAC Receiver Address Register

Table 724 • MAC_EDAC_RX_ADR

Bit Number	Name	Reset Value	Description
[31:26]	Reserved	0	
[25:13]	MAC_EDAC_RX_2E_AD	0	Stores the address from Ethernet RX memory on which a 2-bit SECEDED error has occurred.
[12:0]	MAC_EDAC_RX_1E_AD	0	Stores the address from Ethernet RX memory on which a 1-bit SECEDED error has occurred.

22.3.68 MAC EDAC Transmitter Address Register

Table 725 • MAC_EDAC_TX_ADR

Bit Number	Name	Reset Value	Description
[31:26]	Reserved	0	
[25:13]	MAC_EDAC_TX_2E_AD	0	Stores the address from Ethernet TX memory on which a 2-bit SECEDED error has occurred.
[12:0]	MAC_EDAC_TX_1E_AD	0	Stores the address from Ethernet TX memory on which a 1-bit SECEDED error has occurred.

22.3.69 CAN EDAC Address Register

Table 726 • CAN_EDAC_ADR

Bit Number	Name	Reset Value	Description
[31:26]	Reserved	0	
[25:13]	CAN_EDAC_2E_AD	0	Stores the address from CAN memory on which a 2-bit SECEDED error has occurred.
[12:0]	CAN_EDAC_1E_AD	0	Stores the address from CAN memory on which a 1-bit SECEDED error has occurred.

22.3.70 USB EDAC Address Register

Table 727 • USB_EDAC_ADR

Bit Number	Name	Reset Value	Description
[31:26]	Reserved	0	
[25:13]	USB_EDAC_2E_AD	0	Stores the address from USB memory on which a 2-bit SECEDED error has occurred.
[12:0]	USB_EDAC_1E_AD	0	Stores the address from USB memory on which a 1-bit SECEDED error has occurred.

22.3.71 Security Configuration Register for Masters 0, 1, and 2

Table 728 • MM0_1_2_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	
9	MM0_1_2_MS6_ALLOWED_W	1	Write security bits for masters 0, 1, and 2 to slave 6 (MSS DDR bridge). If not set, masters 0, 1, and 2 will not have write access to slave 6.
8	MM0_1_2_MS6_ALLOWED_R	1	Read security bits for masters 0, 1, and 2 to slave 6 (MSS DDR bridge). If not set, masters 0, 1, and 2 will not have read access to slave 6.
7	MM0_1_2_MS3_ALLOWED_W	1	Write security bits for masters 0, 1, and 2 to slave 3 (eNVM1). If not set, masters 0, 1, and 2 will not have write access to slave 3.
6	MM0_1_2_MS3_ALLOWED_R	1	Read security bits for masters 0, 1 and 2 to slave 3 (eNVM1). If not set, masters 0, 1, and 2 will not have read access to slave 3.
5	MM0_1_2_MS2_ALLOWED_W	1	Write security bits for masters 0, 1, and 2 to slave 2 (eNVM0). If not set, masters 0, 1, and 2 will not have write access to slave 2.
4	MM0_1_2_MS2_ALLOWED_R	1	Read security bits for masters 0, 1, and 2 to slave 2 (eNVM0). If not set, masters 0, 1, and 2 will not have read access to slave 2.
3	MM0_1_2_MS1_ALLOWED_W	1	Write security bits for masters 0, 1, and 2 to slave 1 (eSRAM1). If not set, masters 0, 1, and 2 will not have write access to slave 1.
2	MM0_1_2_MS1_ALLOWED_R	1	Read security bits for masters 0, 1, and 2 to slave 1 (eSRAM1). If not set, masters 0, 1, and 2 will not have read access to slave 1.
1	MM0_1_2_MS0_ALLOWED_W	1	Write security bits for masters 0, 1, and 2 to slave 0 (eSRAM0). If not set, masters 0, 1, and 2 will not have write access to slave 0.
0	MM0_1_2_MS0_ALLOWED_R	1	Read security bits for masters 0, 1, and 2 to slave 0 (eSRAM0). If not set, masters 0, 1, and 2 will not have read access to slave 0.

22.3.72 Security Configuration Register for Masters 4, 5, and DDR_FIC

Table 729 • MM4_5_DDR_FIC_SECURITY/MM4_5_FIC64_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	
9	MM4_5_DDR_FIC_MS6_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 6 (MSS DDR bridge). If not set, masters 4, 5 and DDR_FIC will not have write access to slave 6.
8	MM4_5_DDR_FIC_MS6_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 6 (MSS DDR bridge). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 6.

Table 729 • MM4_5_DDR_FIC_SECURITY/MM4_5_FIC64_SECURITY (continued)

Bit Number	Name	Reset Value	Description
7	MM4_5_DDR_FIC_MS3_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 3 (eNVM1). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 3.
6	MM4_5_DDR_FIC_MS3_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 3 (eNVM1). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 3.
5	MM4_5_DDR_FIC_MS2_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 2 (eNVM0). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 2.
4	MM4_5_DDR_FIC_MS2_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 2 (eNVM0). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 2.
3	MM4_5_DDR_FIC_MS1_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 1 (eSRAM1). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 1.
2	MM4_5_DDR_FIC_MS1_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 1 (eSRAM1). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 1.
1	MM4_5_DDR_FIC_MS0_ALLOWED_W	1	Write security bits for masters 4, 5, and DDR_FIC to slave 0 (eSRAM0). If not set, masters 4, 5, and DDR_FIC will not have write access to slave 0.
0	MM4_5_DDR_FIC_MS0_ALLOWED_R	1	Read security bits for masters 4, 5, and DDR_FIC to slave 0 (eSRAM0). If not set, masters 4, 5, and DDR_FIC will not have read access to slave 0.

22.3.73 Security Configuration Register for Masters 3, 6, 7, and 8

Table 730 • MM3_6_7_8_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	
9	MM3_6_7_8_MS6_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 6 (MSS DDR bridge). If not set, masters 3, 6, 7, and 8 will not have write access to slave 6.
8	MM3_6_7_8_MS6_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 6 (MSS DDR bridge). If not set, masters 3, 6, 7, and 8 will not have read access to slave 6.
7	MM3_6_7_8_MS3_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 3 (eNVM1). If not set, masters 3, 6, 7, and 8 will not have write access to slave 3.
6	MM3_6_7_8_MS3_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 3 (eNVM1). If not set, masters 3, 6, 7, and 8 will not have read access to slave 3.
5	MM3_6_7_8_MS2_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 2 (eNVM0). If not set, masters 3, 6, 7, and 8 will not have write access to slave 2.

Table 730 • MM3_6_7_8_SECURITY (continued)

4	MM3_6_7_8_MS2_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 2 (eNVM0). If not set, masters 3, 6, 7, and 8 will not have read access to slave 2.
3	MM3_6_7_8_MS1_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 1 (eSRAM1). If not set, masters 3, 6, 7, and 8 will not have write access to slave 1.
2	MM3_6_7_8_MS1_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 1 (eSRAM1). If not set, masters 3, 6, 7, and 8 will not have read access to slave 1.
1	MM3_6_7_8_MS0_ALLOWED_W	1	Write security bits for masters 3, 6, 7, and 8 to slave 0 (eSRAM0). If not set, masters 3, 6, 7, and 8 will not have write access to slave 0.
0	MM3_6_7_8_MS0_ALLOWED_R	1	Read security bits for masters 3, 6, 7, and 8 to slave 0 (eSRAM0). If not set, masters 3, 6, 7, and 8 will not have read access to slave 0.

22.3.74 Security Configuration Register for Master 9

Table 731 • MM9_SECURITY

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	
9	MM9_MS6_ALLOWED_W	1	Write security bits for master 9 to slave 6 (MSS DDR bridge). If not set, master 9 will not have write access to slave 6.
8	MM9_MS6_ALLOWED_R	1	Read security bits for master 9 to slave 6 (MSS DDR bridge). If not set, master 9 will not have read access to slave 6.
7	MM9_MS3_ALLOWED_W	1	Write security bits for master 9 to slave 3 (eNVM1). If not set, master 9 will not have write access to slave 3.
6	MM9_MS3_ALLOWED_R	1	Read security bits for master 9 to slave 3 (eNVM1). If not set, master 9 will not have read access to slave 3.
5	MM9_MS2_ALLOWED_W	1	Write security bits for master 9 to slave 2 (eNVM0). If not set, master 9 will not have write access to slave 2.
4	MM9_MS2_ALLOWED_R	1	Read security bits for master 9 to slave 2 (eNVM0). If not set, master 9 will not have read access to slave 2.
3	MM9_MS1_ALLOWED_W	1	Write security bits for master 9 to slave 1 (eSRAM1). If not set, master 9 will not have write access to slave 1.
2	MM9_MS1_ALLOWED_R	1	Read security bits for master 9 to slave 1 (eSRAM1). If not set, master 9 will not have read access to slave 1.
1	MM9_MS0_ALLOWED_W	1	Write security bits for master 9 to slave 0 (eSRAM0). If not set, master 9 will not have write access to slave 0.
0	MM9_MS0_ALLOWED_R	1	Read security bits for master 9 to slave 0 (eSRAM0). If not set, master 9 will not have read access to slave 0.

22.3.75 M3 Status Register

Table 732 • M3_SR

Bit Number	Name	Reset Value	Description
[31:8]	Reserved	0	
7:0	CURRPRI	0	Indicates which priority interrupt, or base boost, is being used now. CURRPRI represents the pre-emption priority, and does not indicate secondary priority.

22.3.76 ETM Count Low Register

Table 733 • ETM_COUNT_LOW

Bit Number	Name	Reset Value	Description
[31:0]	ETMCOUNT_31_0	0	Indicates the 32-bit lower bits of timestamp value (TSVALUEB) from the Cortex-M3 processor.

22.3.77 ETM Count High Register

Table 734 • ETM_COUNT_HIGH

Bit Number	Name	Reset Value	Description
[31:28]	Reserved	0	
[27:25]	ETMINTSTAT	0	Indicates the interrupt status. The following bit definitions mark the interrupt status of the current cycle: 000: No status 001: Interrupt entry 010: Interrupt exit 011: Interrupt return 100: Vector fetch and stack push ETMINTSTAT entry or return is asserted in the first cycle of the new interrupt context. Exit occurs without ETMIVALID.
[24:16]	ETMINTNUM	0	Marks the interrupt number of current execution context.
[15:0]	ETMCOUNT_47_32	0	Indicates the 47 to 32 of timestamp value (TSVALUEB) from the Cortex-M3 processor.

22.3.78 Device Status Register

Table 735 • DEVICE_SR

Bit Number	Name	Reset Value	Description
[31:7]	Reserved	0	
6	M3_DEBUG_ENABLE	0x1	Enables the debug access port (DAP) logic within the Cortex-M3 processor. The reset signal for this bit is SYSRESET_N. The read type is RO-U for this bit. This bit has the following meanings: 0: Debug block of Cortex-M3 is disabled and it is not possible to use a debugger to debug user firmware. 1: Debug block of Cortex-M3 is enabled and it is possible to use a debugger to debug user firmware.
5	M3_DISABLE	0	Disables/enables the Cortex-M3 processor. When this bit is 1, the Cortex-M3 processor is reset. When this is 0, the Cortex-M3 processor will be out of reset.
4	FLASH_VALID_SYNC	0	Asserted when FPGA fabric is valid. There is no reset signal for this bit. This bit has the following meanings: 0: FPGA fabric flash bits are valid and operational 1: FPGA fabric flash bits are not operational
3	WATCHDOG_FREEZE_SYNC	0	Freezes the watchdog counter. There is no reset signal for this bit. This bit has the following meanings: 0: Watchdog counter is not frozen 1: Watchdog counter is frozen (not counting down)
2	FF_IN_PROGRESS_SYNC	0	Indicates the FF_IN_PROGRESS STATE. There is no reset signal for this bit.
1	VIRGIN_PART	0x1	Indicates the device as virgin or non-virgin type. There is no reset signal for this bit. This bit has the following meanings: 0: Device is not a virgin part. It has been through a programming cycle to at least configure the factory settings 1: Device is a virgin part. It has never been through any programming cycle in and all internal flash bits are invalid
0	CORE_UP_SYNC	0	Indicates the status of the synchronized CORE_UP input from the system controller. There is no reset signal for this bit.

22.3.79 eNVM Protect User Register

Table 736 • ENVM_PROTECT_USER

Bit Number	Name	Reset Value	Description
[31:16]	Reserved	0	
15	NVM1_UPPER_WRITE_ALLOWED	0x1	When set, indicates that the masters who have read access can have write access to the upper protection region of eNVM1. This is updated by the user flash row bit.
14	NVM1_UPPER_OTHERS_ACCESS	0x1	When set, indicates that the other masters can access the upper protection region of eNVM1. This is set by the user flash row bit.

Table 736 • ENVM_PROTECT_USER (continued)

Bit Number	Name	Reset Value	Description
13	NVM1_UPPER_FABRIC_ACCESS	0x1	When set, indicates that the fabric can access the upper protection region of eNVM1. This is set by the user flash row bit.
12	NVM1_UPPER_M3ACCESS	0x1	When set, indicates that the Cortex-M3 processor can access the upper protection region of eNVM1. This is updated by the user flash row bit.
11	NVM1_LOWER_WRITE_ALLOWED	0x1	When set, indicates that the masters who have read access can have write access to the lower protection region of eNVM1. This is set by the user flash row bit.
10	NVM1_LOWER_OTHERS_ACCESS	0x1	When set, indicates that the other masters can access the lower protection region of eNVM1. This is set by the user flash row bit.
9	NVM1_LOWER_FABRIC_ACCESS	0x1	When set, indicates that the fabric can access the lower protection region of eNVM1. This will be set by user flash row bit.
8	NVM1_LOWER_M3ACCESS	0x1	When set, indicates that the M3 can access the lower protection region of eNVM1. This will be set by the user flash row bit.
7	NVM0_UPPER_WRITE_ALLOWED	0x1	When set, indicates that the masters who have read access can have write access to the upper protection region of eNVM0. This will be set by the user flash row bit.
6	NVM0_UPPER_OTHERS_ACCESS	0x1	When set, indicates that the other masters can access the upper protection region of eNVM0.
5	NVM0_UPPER_FABRIC_ACCESS	0x1	When set, indicates that the fabric can access the upper protection region of eNVM0. This will be set by the user flash row bit.
4	NVM0_UPPER_M3ACCESS	0x1	When set, indicates that the M3 can access the upper protection region of eNVM0. This will be set by the user flash row bit.
3	NVM0_LOWER_WRITE_ALLOWED	0x1	When set, indicates that the masters who have read access can have write access to the lower protection region of eNVM0. This will be set by the user flash row bit.
2	NVM0_LOWER_OTHERS_ACCESS	0x1	When set, indicates that the other masters can access the lower protection region of eNVM0. This will be set by the user flash row bit.
1	NVM0_LOWER_FABRIC_ACCESS	0x1	When set, indicates that the fabric can access the lower protection region of eNVM0. This will be set by the user flash row bit.
0	NVM0_LOWER_M3ACCESS	0x1	When set, indicates that the M3 can access the lower protection region of eNVM0. This will be set by the user flash row bit.

22.3.80 Smart Fusion2 eNVM Status Register

Table 737 • ENVM_STATUS

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	Reserved
0	CODE_SHADOW_EN	0	Read by the System Controller during device start-up, to indicate whether the user has configured the device such that code shadowing is to be performed by system controller firmware.

22.3.81 Device Version Register

Table 738 • DEVICE_VERSION

Bit Number	Name	Reset Value	Description
[31:20]	Reserved	0	
[19:16]	IDV	0	Internal device version.
[15:0]	IDP	0	Internal device product.

22.3.82 MSS DDR PLL Status Register

Table 739 • MSSDDR_PLL_STATUS

Bit Number	Name	Reset Value	Description
[31:3]	Reserved	0	
2	RCOSC_DIV2		Input from the System Controller, indicating whether the 50 MHz RC oscillator is running at 25 MHz or 50 MHz. 0: Running at 25MHz 1: Running at 50MHz
1	MPLL_LOCK	0	MPLL lock status. A LOCK signal is provided to indicate that the MPLL has locked on to the incoming signal. LOCK asserts High to indicate that the MPLL has achieved frequency and phase lock. Allowed values are: 0: MPLL is not in lock 1: MPLL is in lock Microsemi recommends that LOCK is only used for test and system status information, and is not used for critical system functions without thorough characterization in the host system. The precision of the LOCK discrimination can be adjusted using the LOCKWIN[2:0] controls. The integration of the LOCK period can be adjusted using the LOCKCNT[3:0] controls.

Table 739 • MSSDDR_PLL_STATUS (continued)

Bit Number	Name	Reset Value	Description
0	FAB_PLL_LOCK	0	<p>If CLK_BASE is generated from a PLL in the fabric, this signal must be connected from the LOCK output of that PLL. When the FACC is going through its PLL initialization stage (either under system controller control or MSS master control), this signal is ANDed with the LOCK output of the MPLL. Only when both PLLs are in lock, is the system considered to be ready for switching to PLL-derived clock. If CLK_BASE is not derived from a fabric PLL, then the user must ensure that this signal is tied High at the fabric interface. Allowed values:</p> <p>0: Fabric PLL is not in lock. 1: Fabric PLL is in lock or CLK_BASE is not derived from a fabric PLL.</p>

22.3.83 USB Status Register

Table 740 • USB_SR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	
1	LPI_CARKIT_EN	0	Asserted when entry is made into CarKit mode and cleared on exit from CarKit mode.
0	POWERDN	0	Asserted when CLK may be stopped to save power.

22.3.84 eNVM Status Register

Table 741 • ENVM_SR

Bit Number	Name	Reset Value	Description
[31:2]	Reserved	0	
1:0	ENVM_BUSY	0	<p>Active high signals indicate a busy state per eNVM for CLK-driven operations and for internal operations triggered by the write/program/erase/transfer command.</p> <p>ENVM_BUSY[1] = Busy indication from eNVM1 ENVM_BUSY[0] = Busy indication from eNVM0</p>

22.3.85 DDRB Status Register

Table 742 • DDRB_STATUS

Bit Number	Name	Reset Value	Description
[31:0]	DDRB_DEBUG_STATUS	0x1	<p>Status of the internal ports of DDRBRIDGE. The bit definitions are as follows:</p> <p>Debug ports of the MSS DDR bridge:</p> <p>SYR_DDRB_DP[31:30] = DSG write buffer mode status</p> <p>SYR_DDRB_DP[29:28] = AHB bus write buffer mode status</p> <p>SYR_DDRB_DP[27:26] = HPDMA write buffer mode status</p> <p>SYR_DDRB_DP[25:23] = IDC read buffer mode status</p> <p>SYR_DDRB_DP[22:20] = DSG read buffer mode status</p> <p>SYR_DDRB_DP[19:17] = AHB bus read buffer mode status</p> <p>SYR_DDRB_DP[16:14] = HPDMA read buffer mode status</p> <p>SYR_DDRB_DP[13] = DSG write request to arbiter</p> <p>SYR_DDRB_DP[12] = AHB bus write request to arbiter</p> <p>SYR_DDRB_DP[11] = HPDMA write request to arbiter</p> <p>SYR_DDRB_DP[10] = IDC read req to arbiter</p> <p>SYR_DDRB_DP[9] = DSG read req to arbiter</p> <p>SYR_DDRB_DP[8] = AHB bus read req to arbiter</p> <p>SYR_DDRB_DP[7] = HPDMA read request to arbiter</p> <p>SYR_DDRB_DP[6] = AXI write address channel acknowledge to DSG write request</p> <p>SYR_DDRB_DP[5] = AXI write address channel acknowledge to AHB bus write request</p> <p>SYR_DDRB_DP[4] = AXI write address channel acknowledge to HPDMA write request</p> <p>SYR_DDRB_DP[3] = AXI write data channel acknowledge to DSG write request</p> <p>SYR_DDRB_DP[2] = AXI write data channel acknowledge to AHB bus write request</p> <p>SYR_DDRB_DP[1] = AXI write data channel acknowledge to HPDMA write request</p> <p>SYR_DDRB_DP[0] = Lock input to arbiter from AHB bus WCB</p>

22.3.86 MDDR IO Calibration Status Register

Table 743 • MDDR_IO_CALIB_STATUS

Bit Number	Name	Reset Value	Description
[31:15]	Reserved	0	
14	CALIB_PCOMP	0x1	State of the P analog comparator
13	CALIB_NCOMP	0x1	State of the N analog comparator
[12:6]	CALIB_PCODE	0x3F	Current PCODE value set on the MDDR DDR I/O bank
[5:1]	CALIB_NCODE	0x3F	Current NCODE value set on the MDDR DDR I/O bank
0	CALIB_STATUS	0	1 when the codes are actually locked. For the first run after reset, this would be asserted 1 cycle after CALIB_INTRPT. For in-between runs, this would be asserted only when the DRAM is put into self-refresh or there is an override from the firmware (CALIB_LOCK).

22.3.87 MSS DDR Clock Calibration Status

Table 744 • MSSDDR_CLK_CALIB_STATUS

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	FAB_CALIB_FAIL	0	0: The currently selected CCC delay values for the M3_CLK and fabric Clock are such that the FPGA fabric clock calibration circuit is running correctly. 1: The FPGA fabric clock calibration circuit is failing to operate correctly. This indicates incorrectly configured delay values for M3_CLK and/or fabric clock in the CCC.

22.3.88 Watch Dog Load Register

Table 745 • WDOGLOAD

Bit Number	Name	Reset Value	Description
[31:26]	Reserved	0	
[25:0]	WDOGLOAD	0x1800000	Contains upper 26 bits of the WDOGLOAD value register

22.3.89 Watch Dog MVRP Register

Table 746 • WDOGMVRP

Bit Number	Name	Reset Value	Description
[31:0]	WDOGMVRP	0xFFFFFFFF	Contains the WDOGMVRP value

22.3.90 User Configuration Register 0

Table 747 • USERCONFIG0

Bit Number	Name	Reset Value	Description
[31:0]	CONFIG_REG0	0	Stores the user configuration register 0 to be read by the Cortex-M3 processor.

22.3.91 User Configuration Register 1

Table 748 • USERCONFIG1

Bit Number	Name	Reset Value	Description
[31:0]	CONFIG_REG1	0	Stores the user configuration register 1 to be read by the Cortex-M3 processor.

22.3.92 User Configuration Register 2

Table 749 • USERCONFIG2

Bit Number	Name	Reset Value	Description
[31:0]	CONFIG_REG2	0	Stores the user configuration register 2 to be read by the Cortex-M3 processor.

22.3.93 User Configuration Register 3

Table 750 • USERCONFIG3

Bit Number	Name	Reset Value	Description
[31:0]	CONFIG_REG3	0	Stores the user configuration register 3 to be read by the Cortex-M3 processor.

22.3.94 Fabric Protected Size Register

Table 751 • FAB_PROT_SIZE

Bit Number	Name	Reset Value	Description
[31:6]	Reserved	0	
[5:0]	SW_PROTREGIONSIZE	11110	The size of the memory region inaccessible to the FPGA fabric master is determined by the value of this bus. The region sizes are listed in Table 752 , page 728.

22.3.94.1 Region Size

Table 752 • Region Size

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Size
0	0	0	0	0	Reserved
0	0	0	0	1	Reserved
0	0	0	1	0	Reserved
0	0	0	1	1	Reserved
0	0	1	0	0	Reserved
0	0	1	0	1	Reserved
0	0	1	1	0	128 Bytes
0	0	1	1	1	Reserved
0	1	0	0	0	Reserved
0	1	0	0	1	Reserved
0	1	0	1	0	2 KB
0	1	0	1	1	Reserved

Table 752 • Region Size (continued)

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Size
0	1	1	0	0	Reserved
0	1	1	0	1	16 KB
0	1	1	1	0	32 KB
0	1	1	1	1	64 KB
1	0	0	0	0	128 KB
1	0	0	0	1	256 KB
1	0	0	1	0	512 KB
1	0	0	1	1	Reserved
1	0	1	0	0	Reserved
1	0	1	0	1	Reserved
1	0	1	1	0	8 Mbytes
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved

22.3.95 Fabric Protected Base Address Register

Table 753 • FAB_PROT_BASE

Bit Number	Name	Reset Value	Description
[31:0]	SW_PROTREGIONBASE	0	<p>The base address of the memory region inaccessible to the FPGA fabric master is determined by the value of this bus. Bit 0 of this bus is defined as SW_PROTREGIONENABLE. This has the following meaning:</p> <p>0: Protected region not enabled. This means that a master in the FPGA fabric may access any location in the memory map, as long as the fabric master port is enabled.</p> <p>1: Protected region enabled. Any accesses attempted by a fabric master to this region of memory return an error in the bus transaction.</p> <p>Bits [31:N] of this bus indicate the base address of the protected region.</p>

The value of N depends on the protected region size, so that the base address is aligned according to an even multiple of region size. The power of 2 size specified by SW_PROTREGIONSIZ[4:0] defines how many bits of base address are used. For example, if the SW_PROTREGIONSIZ[4:0] is 01111, this corresponds to a protected region of 64 KB. 64 KB is 2 to the power of 16. Therefore the value of N in this case is 16. So the base address of the region, in this case, is specified by SW_PROTREGIONBASE[31:16].

22.3.96 MSS GPIO Definitions

Table 754 • MSS_GPIO_DEF

Bit Number	Name	Reset Value	Description
[31:4]	Reserved	0	
3	MSS_GPIO_31_24_DEF	0x1	Used to initialize GPIO Bank [31:24] to 0 or 1 after reset.
2	MSS_GPIO_23_16_DEF	0x1	Used to initialize GPIO Bank [23:16] to 0 or 1 after reset.
1	MSS_GPIO_15_8_DEF	0x1	Used to initialize GPIO Bank [15:8] to 0 or 1 after reset.
0	MSS_GPIO_7_0_DEF	0x1	Used to initialize GPIO Bank [7:0] to 0 or 1 after reset.

22.3.97 EDAC Status Register

Table 755 • EDAC_SR

Bit Number	Name	Reset Value	Description
[31:14]	Reserved	0	
13	CAN_EDAC_2E	0	Updated by CAN when a 2-bit SECEDED error has been detected for RAM memory.
12	CAN_EDAC_1E	0	Updated by CAN when a 1-bit SECEDED error has been detected and is corrected for RAM memory.
11	USB_EDAC_2E	0	Updated by USB when a 2-bit SECEDED error has been detected for RAM memory.
10	USB_EDAC_1E	0	Updated by USB when a 1-bit SECEDED error has been detected and is corrected for RAM memory.
9	MAC_EDAC_RX_2E	0	Updated by Ethernet when a 2-bit SECEDED error has been detected for Rx RAM memory.
8	MAC_EDAC_RX_1E	0	Updated by Ethernet when a 1-bit SECEDED error has been detected and is corrected for Rx RAM memory.
7	MAC_EDAC_TX_2E	0	Updated by Ethernet when a 2-bit SECEDED error has been detected for Tx RAM memory.
6	MAC_EDAC_TX_E	0	Updated by Ethernet when a 1-bit SECEDED error has been detected and is corrected for Tx RAM memory.
5	Reserved	0	
4	Reserved	0	
3	ESRAM1_EDAC_2E	0	Updated by the eSRAM_1 controller when a 2-bit SECEDED error has been detected for eSRAM1 memory.
2	ESRAM1_EDAC_1E	0	Updated by the eSRAM_1 Controller when a 1-bit SECEDED error has been detected and is corrected for eSRAM1 memory.
1	ESRAM0_EDAC_2E	0	Updated by the eSRAM_0 controller when a 2-bit SECEDED error has been detected for eSRAM0 memory.
0	ESRAM0_EDAC_1E	0	Updated by the eSRAM_0 controller when a 1-bit SECEDED error has been detected and is corrected for eSRAM0 memory.

22.3.98 MSS Internal Status Register

Table 756 • MSS_INTERNAL_SR

Bit Number	Name	Reset Value	Description
[31:7]	Reserved	0	
6	DDR_FIC_INT	0	Indicates an interrupt from DDR_FIC.
5	MDDR_ECC_INT	0	Indicates when an SECEDED interrupt from the MDDR subsystem is asserted.
4	MDDR_IO_CALIB_INT	0	Interrupt is generated when the MDDR calibration is finished.
3	FAB_PLL_LOCKLOST_INT	0	Indicates that a falling edge event occurred on the FAB_PLL_LOCK signal. This indicates that the fabric PLL lost lock.
2	FAB_PLL_LOCK_INT	0	Indicates that a rising edge event occurred on the FAB_PLL_LOCK signal. This indicates that the fabric PLL came into lock.
1	MPLL_LOCKLOST_INT	0	Indicates that a falling edge event occurred on the MPLL_LOCK signal. This indicates that the MPLL lost lock.
0	MPLL_LOCK_INT	0	Indicates that a rising edge event occurred on the MPLL_LOCK signal. This indicates that the MPLL came into lock.

22.3.99 MSS External Status Register

Table 757 • MSS_EXTERNAL_SR

Bit Number	Name	Reset Value	Description
[31:19]	Reserved	0	
18	CC_HRESP_ERR		Indicates whether any accesses to the corresponding master on the CACHE resulted in HRESP assertion by the slave to the CACHE (and hence to the master) or HRESP generated by the CACHE itself to the master (in the case of an invalid address being accessed). The CACHE does pass the HRESP signal through to the requesting master, but the event is also registered in this register. The bit definitions are as follows: Bit 0: Corresponds to an HRESP assertion being issued to the DCode bus master. Bit 1: Corresponds to an HRESP assertion being issued to the ICode bus master. Bit 2: Corresponds to an HRESP assertion being issued to the SBus master.
17	DDRB_LOCK_MID	0	Indicates which master (AHB bus or HPDMA) is responsible for lock timeout condition. 0: AHB bus master 1: HPDMA

Table 757 • MSS_EXTERNAL_SR (continued)

Bit Number	Name	Reset Value	Description
16	DDRB_LCKOUT	0	Asserted when lock timeout counter reaches its maximum value. Lock time out counter (20-bit) is maintained in the MSS DDR bridge, which starts counting when a locked transfer obtains access to the AXI bus. When the counter reaches maximum value, a DDRB_LCKOUT interrupt is generated and stays asserted until cleared by the processor.
15	DDRB_HPD_WR_ERR	0	Asserted when the MSS DDR bridge gets an error response from the DDR slave for an HPDMA write request. Address of write transaction for which error response is received is provided by DDRB_HPD_ERR_ADD.
14	DDRB_SW_WR_ERR	0	Asserted when the MSS DDR bridge gets an error response from the DDR slave for an AHB bus master write request. Address of write transaction for which error response is received is provided by DDRB_SW_ERR_ADD.
13	DDRB_DS_WR_ERR	0	Asserted when the MSS DDR bridge gets an error response from the DDR slave for a DS master write request. Address of write transaction for which error response is received is provided by DDRB_DS_ERR_ADD.
[12:7]	DDRB_RDWR_ERR_REG	0	Provides the read/write address match error status generated during the following accesses: Bit 0 = 1: AHB bus and HPDMA are trying to access same address Bit 1 = 1: AHB bus and DS are trying to access same address Bit 2 = 1: HPDMA and DS are trying to access same address Bit 3 = 1: IDC and HPDMA are trying to access same address Bit 4 = 1: IDC and AHB Bus are trying to access same address Bit 5 = 1: IDC and DS are trying to access same address
[6:0]	SW_ERRORSTATUS	0	Indicates whether any accesses by the corresponding master on the AHB bus resulted in either HRESP assertion by the slave to the AHB bus, HRESP assertion by the AHB bus to that master (in the case of blocked fabric master), or was decoded by the AHB bus as being to “unimplemented” address space. The bit definitions are as follows: Bit 0: Corresponds to an HRESP assertion being issued to the HPDMA interface Bit 1: Corresponds to an HRESP assertion being issued to FIC_0 interface Bit 2: Corresponds to an HRESP assertion being issued to FIC_1 System interface S interface Bit 3: Corresponds to an HRESP assertion being issued to the Ethernet MAC Bit 4: Corresponds to an HRESP assertion being issued to the peripheral DMA engine Bit 5: Corresponds to an HRESP assertion being issued to the USB Bit 6: Corresponds to an HRESP assertion being issued to the System Controller These signals are not used as interrupts to the Cortex-M3 processor. Instead, they are ORed together in SYSREG to create a signal called SW_ERRORINTERRUPT, which is used as an interrupt to the Cortex-M3 processor.

22.3.100 Watchdog Timeout Event

Table 758 • WDOGTIMEOUTEVENT

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	WDOGTIMEOUTEVENT	0	WDOGTIMEOUTEVENT is not affected by SYSRESETN. This allows firmware to determine if a system reset occurred due to a watchdog timeout event. This signal is not used as an interrupt to the Cortex-M3 processor. Reset to 0 by PORESETN only.

22.3.101 Clear MSS Counters

Table 759 • CLR_MSS_COUNTERS

Bit Number	Name	Reset Value	Description
[31:6]	Reserved	0	
5	CC_DC_TRANS_CNTCLR	0	When set, the CC_DC_TRANS_CNT counter is reset.
4	CC_IC_TRANS_CNTCLR	0	When set, the CC_IC_TRANS_CNT counter is reset.
3	CC_DC_HIT_CNTCLR	0	When set, the CC_DC_HIT_CNT counter is reset.
2	CC_DC_MISS_CNTCLR	0	When set, the CC_DC_MISS_CNT counter is reset.
1	CC_IC_HIT_CNTCLR	0	When set, the CC_IC_HIT_CNT counter is reset.
0	CC_IC_MISS_CNTCLR	0	When set, the CC_IC_MISS_CNT counter is reset.

22.3.102 Clear EDAC Counters

Table 760 • CLR_EDAC_COUNTERS

Bit Number	Name	Reset Value	Description
[31:14]	Reserved	0	
13	CAN_EDAC_CNTCLR_2E	0	Pulse generated to clear the 16-bit counter value in CAN corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the CAN_EDAC_CNT register.
12	CAN_EDAC_CNTCLR_1E	0	Pulse generated to clear the 16-bit counter value in CAN corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the CAN_EDAC_CNT register.
11	USB_EDAC_CNTCLR_2E	0	Pulse generated to clear the 16-bit counter value in USB corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the USB_EDAC_CNT register.
10	USB_EDAC_CNTCLR_1E	0	Pulse generated to clear the 16-bit counter value in USB corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the USB_EDAC_CNT register.
9	MAC_EDAC_RX_CNTCLR_2E	0	Pulse generated to clear the 16-bit counter value in Ethernet MAC Rx RAM corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the MAC_EDAC_RX_CNT register.

Table 760 • CLR_EDAC_COUNTERS (continued)

Bit Number	Name	Reset Value	Description
8	MAC_EDAC_RX_CNTCLR_1E	0	Pulse generated to clear the 16-bit counter value in Ethernet MAC Rx RAM corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the MAC_EDAC_RX_CNT register.
7	MAC_EDAC_TX_CNTCLR_2E	0	Pulse generated to clear the 16-bit counter value in Ethernet MAC Tx RAM corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the MAC_EDAC_TX_CNT register.
6	MAC_EDAC_TX_CNTCLR_1E	0	Pulse generated to clear the 16-bit counter value in Ethernet MAC Tx RAM corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the MAC_EDAC_TX_CNT register.
5	Reserved	0	
4	Reserved	0	
3	ESRAM1_EDAC_CNTCLR_2E	0	Pulse generated to clear the 16-bit counter value in eSRAM1 corresponding to the count value of EDAC 2-bit errors. This in turn clears the upper 16 bits of the eSRAM1_EDAC_CNT Register.
2	ESRAM1_EDAC_CNTCLR_1E	0	Pulse generated to clear the 16-bit counter value in eSRAM1 corresponding to count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the eSRAM1_EDAC_CNT Register.
1	ESRAM0_EDAC_CNTCLR_2E	0	Pulse generated to clear the 16-bit counter value in eSRAM0 corresponding to count value of EDAC 2bit Errors. This in turn clears the upper 16 bits of eSRAM0_EDAC_CNT the register.
0	ESRAM0_EDAC_CNTCLR_1E	0	Pulse generated to clear the 16-bit counter value in eSRAM0 corresponding to the count value of EDAC 1-bit errors. This in turn clears the lower 16 bits of the ESRAM0_EDAC_CNT Register.

22.3.103 Flush Configuration Register

Table 761 • FLUSH_CR

Bit Number	Name	Reset Value	Description
[31:9]	Reserved	0	
8	DDRB_INVALID_IDC	0	Allows the read buffer for the IDC master in the MSS DDR bridge to be invalidated. The read buffer is emptied once this pulse is detected. 0: No effect 1: Invalidate IDC read buffer
7	DDRB_INVALID_HPD	0	Allows the read buffer allocated for the AHB bus in the MSS DDR bridge to be invalidated. The read buffer is emptied once this pulse is detected. 0: No effect 1: Invalidate HPD read buffer

Table 761 • FLUSH_CR (continued)

Bit Number	Name	Reset Value	Description
6	DDRB_INVALID_SW	0	Allows the read buffer for the high performance master in the MSS DDR bridge to be invalidated. The read buffer is emptied once this pulse is detected. 0: No effect 1: Invalidate AHB Bus read buffer
5	DDRB_INVALID_DS	0	Allows the read buffer for the DSG Master in the MSS DDR bridge to be invalidated. The read buffer is emptied once this pulse is detected. 0: No effect 1: Invalidate DSG read buffer
4	DDRB_FLSHSW	0	Allows the write buffer for the AHB bus in the MSS DDR bridge to be flushed. Data present in the write buffer is transferred to the MSS DDR bridgewrite arbiter interface when this pulse is detected. 0: No effect 1: Flush AHB bus write buffer
3	DDRB_FLSHHPD	0	Allows the write buffer for the HPD master in the MSS DDR bridge to be flushed. Data present in the write buffer is transferred to the MSS DDR bridge write arbiter interface when this pulse is detected. 0: No effect 1: Flush HPD write buffer
2	DDRB_FLSHDS	0	Allows the write buffer for the DSG master in the MSS DDR bridge to be flushed. Data in the write buffer is transferred to the MSS DDR bridge write arbiter when this pulse is detected. 0: No effect 1: Flush DSG write buffer
1	CC_FLUSH_CHLINE	0	Signal (pulse) to flush only one index in the cache memory. This signal is used to invalidate all tags of four sets at one index only. 0: No effect 1: Flush index
0	CC_FLUSH_CACHE	0	Signal (pulse) to flush cache memory. This signal is used to invalidate all tags of four sets at the same time. Allowed values: 0: No effect 1: Flush cache memory

22.3.104 MAC Status Clear Control Register

Table 762 • MAC_STAT_CLR_CR

Bit Number	Name	Reset Value	Description
[31:1]	Reserved	0	
0	MAC_STAT_CLR	0	Writing a '1' to this bit will clear the MAC statistics registers.

22.3.105 IOMUXCELL_CONFIG[n] Configuration Register

Table 763 • IOMUXCELL_CONFIG[n]

Bit Number	Name	Reset Value	Description
[31:10]	Reserved	0	
9	MSS_IOMUXSEL5LOWER[N]	0	Used to select the source of the output port of the I/O cell corresponding to this IOMUXCELL. Each bit of this bus is used together with the corresponding bit of MSS_IOMUXSEL5UPPER and MSS_IOMUXSEL5MID to form a 3-bit field. This field definition is in Table 764 , page 737.
8	MSS_IOMUXSEL5MID[N]	0	Used to select the source of the output port of the I/O cell corresponding to this IOMUXCELL
7	MSS_IOMUXSEL5UPPER[N]	0	Used to select the source of the OE port of the I/O cell corresponding to this IOMUXCELL
6:4	MSS_IOMUXSEL4[N][2:0]	0	Used to select the source of the output port of the I/O cell corresponding to this IOMUXCELL. This field definition is in Table 765 on page 737 .
3	MSS_IOMUXSEL3[N]	0	0: Output enable of interface MSS GPIO of IOMUXCELL goes to FPGA fabric core input 1: Output of interface serial comms of IOMUXCELL goes to FPGA fabric core input
2	MSS_IOMUXSEL2[N]	0	0: Output of interface MSS GPIO of IOMUXCELL goes to FPGA fabric core input 1: Output enable of interface serial comms of IOMUXCELL goes to FPGA fabric core input
1	MSS_IOMUXSEL1[N]	0	0: Input of interface MSS GPIO of IOMUXCELL comes from the output of interface FPGA fabric core 1: Input of interface MSS GPIO of IOMUXCELL comes from the I/O cell input
0	MSS_IOMUXSEL0[N]	0	0: Input of interface serial comms of IOMUXCELL comes from the output of interface FPGA fabric core 1: Input of interface serial comms of IOMUXCELL comes from the I/O cell input

Notes:

- Bit 0, Bit 1, and Bit 2 in the following table refer to IOMUXSEL5LOWER, IOMUXCELL5MID, IOMUXCELL5UPPER respectively.
- Do not change these register fields dynamically for 005 and 010 devices, see ["System Registers Behavior for M2S005/010 Devices"](#) section on page 682.

22.3.105.1MSS_IOMUXSEL5[N][2:0]

Table 764 • MSS_IOMUXSEL5 [N][2:0]

Bit 0	Bit 1	Bit 2	Function
0	0	0	Output of I/O cell comes from output of interface Serial Comms of IOMUXCELL
0	0	1	Output of I/O cell comes from output of interface MSS GPIO of IOMUXCELL
0	1	0	Output of I/O cell comes from output of interface USB Controller of IOMUXCELL
0	1	1	Output of I/O cell is connected to '0'
1	0	0	Output of I/O cell is connected to '1'
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

22.3.105.2MSS_IO_MUXSEL4[N][2:0]

Table 765 • MSS_IOMUXSEL4[N][2:0]

Bit 0	Bit 1	Bit2	Function
0	0	0	Output of I/O cell comes from output of interface Serial Comms of IOMUXCELL
0	0	1	Output of I/O cell comes from output of interface MSS GPIO of IOMUXCELL
0	1	0	Output of I/O cell comes from output of interface USB Controller of IOMUXCELL
0	1	1	Output of I/O cell is connected to '0'
1	0	0	Output of I/O cell is connected to '1'
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

23 Fabric Interface Interrupt Controller

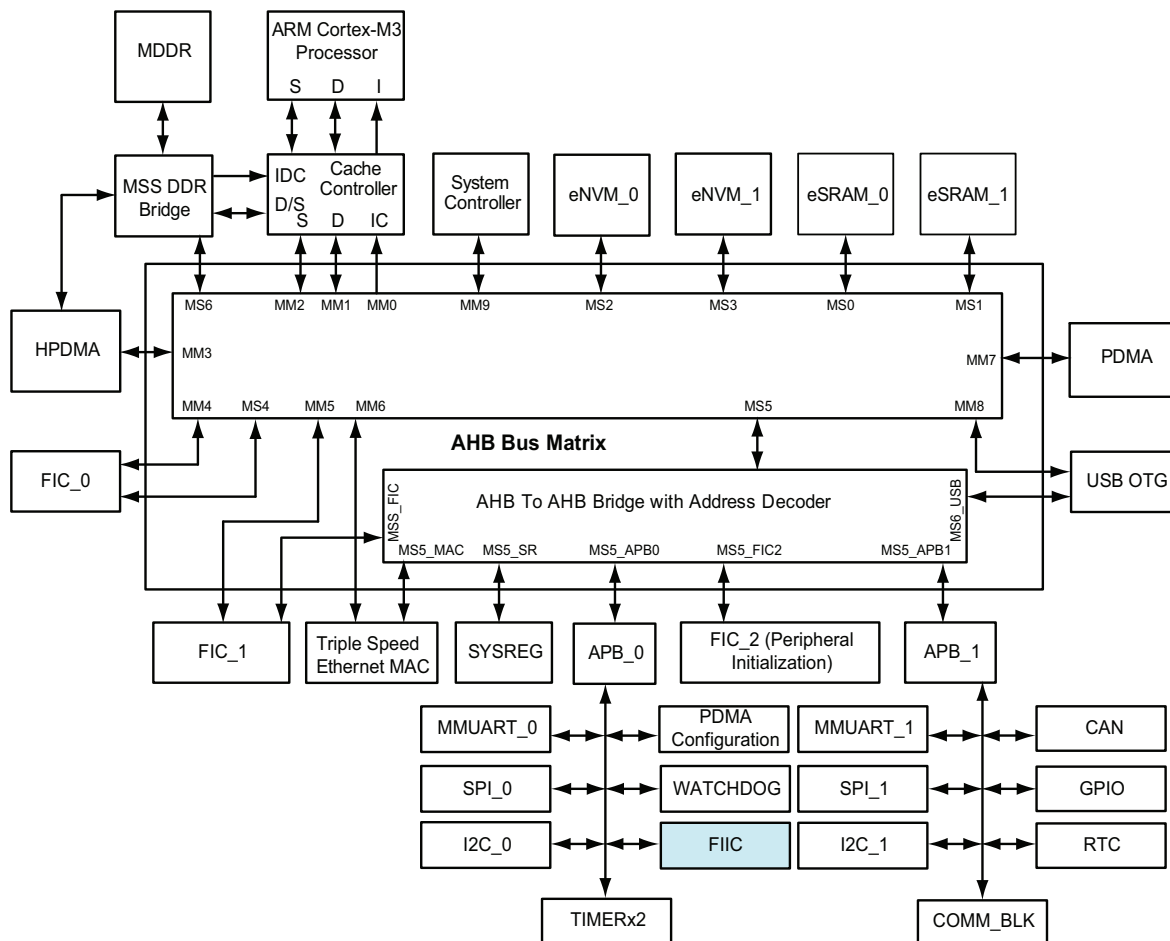
The fabric interface interrupt controller (FIIC) gathers interrupt signals from within the microcontroller subsystem (MSS) and makes them available to the FPGA fabric. There are a number of peripherals and other blocks within the MSS that generate interrupt signals. These interrupt signals are connected to the nested vectored interrupt controller (NVIC) of the Cortex-M3 processor, and can be used as potential interrupt sources to a user logic within the FPGA fabric.

23.1 Features

- FIIC receives 43 interrupts from the MSS as inputs
- 16 individually configurable MSS to fabric interrupt ports
- 16 individually configurable fabric to MSS interrupt ports

The following figure depicts the connectivity of FIIC to the AHB bus matrix.

Figure 319 • The FIIC Connection to AHB Bus Matrix



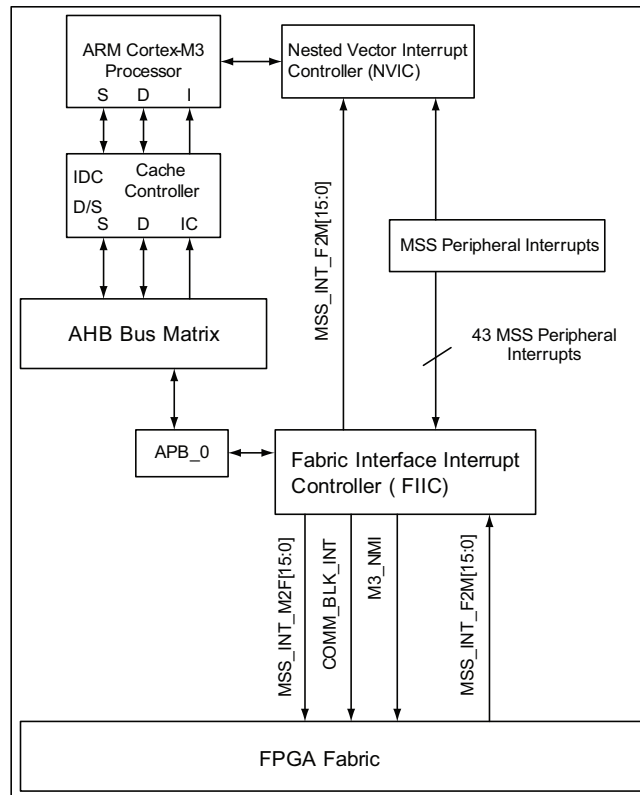
23.2 Functional Description

This section provides the detailed description of the FIIC subsystem.

23.2.1 Architecture Overview

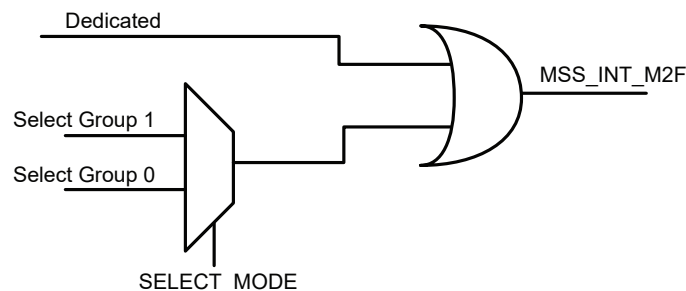
The following figure shows the interfacing of the FIIC with NVIC, MSS peripheral interrupts, and FPGA fabric. The FIIC receives 43 level-sensitive active high interrupts from the MSS as inputs. These MSS peripheral interrupts are combined, in a predetermined fashion, into 16 M2F interrupts (MSS_INT_M2F [15:0]) routed to the fabric. There is also a pass-through M3_NMI non-maskable interrupt from the watchdog timer and COMM_BLK interrupt, COMM_BLK_INT.

Figure 320 • Block Diagram for Fabric Interface Interrupt Controller



There are 16 circuits, as shown in the following figure. Each circuit corresponds to a row in the preceding figure. The dedicated interrupts coming from the MSS peripherals are always connected to the 16 M2F interrupt signals.

Figure 321 • Combinational Circuit for Mapping MSS Interrupts to a MSS_INT_M2F



Every peripheral interrupt in the MSS except I2C_SMBALERT0, I2C_SMBSUS0, I2C_SMBALERT1, and I2C_SMBSUS1, has access to the FPGA fabric through the dedicated inputs of the FIIC.

Each fabric MSS_INT_M2F signal can be triggered from one of the two possible scenarios:

- Dedicated interrupts
- Multiplexed group of interrupts

The selection of the MSS interrupt to a specific MSS_INT_M2F signal and making it available to the FPGA Fabric is done in two stages:

- Select the group of interrupts: It can be done by setting Select_Mode bit of the M2F Interrupt Mode Register.
- Enable the MSS interrupts: It can be done by writing to the appropriate FIIC INTERRUPT_ENABLE0 and INTERRUPT_ENABLE1 interrupt enable registers.

Table 766 • Interrupt Line Signal Distribution

M2F Interrupt Signal	Dedicated	Select Group 0	Select Group 1
MSS_INT_M2F[0]	SPIINT0	ENVM_INT0	HPD_XFR_ERR_INT
MSS_INT_M2F[1]	SPIINT1	ENVM_INT1	MSSDDR_PLL_LOCK_INT
MSS_INT_M2F[2]	I2C_INT0	USB_DMA_INT	SW_ERRORINTERRUPT
MSS_INT_M2F[3]	I2C_INT1	Reserved	DDRB_INTR
MSS_INT_M2F[4]	MMUART0_INTR	I2C_SMBALERT0	ECCINTR
MSS_INT_M2F[5]	MMUART1_INTR	I2C_SMBSUS0	CACHE_ERRINTR
MSS_INT_M2F[6]	MAC_INT	I2C_SMBALERT1	SOFTINTERRUPT
MSS_INT_M2F[7]	USB_MC_INT	I2C_SMBSUS1	COMM_BLK_INTR
MSS_INT_M2F[8]	PDMAINTERRUPT	HPD_XFR_ERR_INT	Reserved
MSS_INT_M2F[9]	HPD_XFR_CMP_INT	MSSDDR_PLL_LOCK_INT	Reserved
MSS_INT_M2F[10]	TIMER1_INTR	SW_ERRORINTERRUPT	Reserved
MSS_INT_M2F[11]	TIMER2_INTR	DDRB_INTR	MDDR_IO_CALIB_INT
MSS_INT_M2F[12]	CAN_INTR	ECCINTR	Reserved
MSS_INT_M2F[13]	RTC_WAKEUP_INTR	CACHE_ERRINTR	FAB_PLL_LOCK_INT
MSS_INT_M2F[14]	WDOGWAKEUPINT	SOFTINTERRUPT	FAB_PLL_LOCKLOST_INT
MSS_INT_M2F[15]	MSSDDR_PLL_LOCKLOST_INT	COMBLK_INTR	FIC64_INT

It is possible to overlay one interrupt signal with two interrupt sources. For example, enable a dedicated interrupt and a group 0/1 interrupt. User logic in the fabric is responsible for determining the actual source of the interrupt by reading the appropriate peripheral interrupt Status Registers and determining which interrupt has occurred. Interrupts In and Out of the FIIC are asynchronous.

All interrupts originating from MSS blocks and fed into the FIIC are active high level sensitive signals. Once asserted, the interrupt remains asserted until the user logic clears the appropriate MSS

peripheral interrupt clear register. MSS_INT_M2F interrupt signals are serviced by the FPGA fabric. The exceptions to this are the SMBALERT and SMBSUS interrupts from the I2C peripheral. When these are held asserted, they are cleared by the far end I2C device, after a firmware-initiated sequence of operations. WDOGTIMEOUTINT is always passed straight through the block as M3_NMI.

F2M interrupts from the fabric are connected to the Cortex-M3 processor NVIC. MSS_INT_F2M [15:0], the 16 F2M interrupts from user logic in the fabric, are routed directly to the Cortex-M3 processor NVIC. F2M interrupts are level sensitive active high inputs.

Once asserted, user logic in the fabric must keep the interrupt asserted until it is cleared by the Cortex-M3 processor firmware. The SmartFusion2 SoC FPGA FIIC does not synchronize fabric sourced peripheral interrupts to the fabric clock or MSS clock.

23.2.2 FIIC Port List

Table 767 • FIIC Port List

Port name	Direction	Polarity	Description
MSS_INT_M2F[15:0]	Out	High	MSS to fabric interrupts. The FIIC routes MSS peripheral interrupts to the fabric.
COMM_BLK_INT	Out		COMMS block interrupt.
M3_NMI	Out		Non-maskable interrupt from watchdog timer.
MSS_INT_F2M[15:0]	In	High	Fabric to MSS interrupts.

23.3 How to Use FIIC

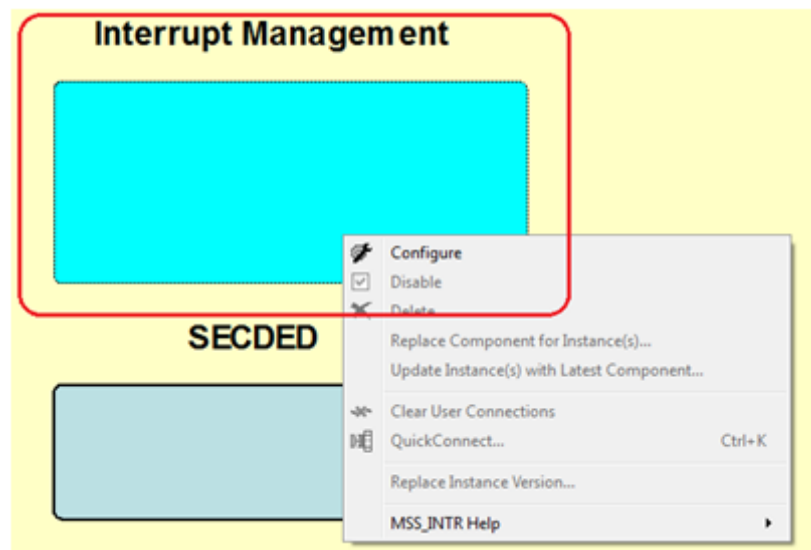
This section describes how to use the FIIC subsystem in the design.

23.3.1 Configuring the FIIC Using the Libero SoC

This section describes the FIIC configuration in the Libero SoC and shows different options available for configuring the FIIC. The FIIC is not configured by default in the MSS configurator when the Libero SoC project is created. The following steps are required to configure in the Libero SoC.

1. Instantiate the SmartFusion2 MSS component into the Libero project and configure (**enable/disable**) the peripherals as per the application needs, using the MSS configurator.
2. Double-click or right-click **Interrupt Management** and select the **configure** option, as shown in the following figure.

Figure 322 • Configure FIIC in the MSS Configurator



The following options are available for configuring the FIIC subsystem.

- Use Fabric to MSS Interrupt
- Use MSS to Fabric Interrupt

The following figure shows the FIIC Configurator.

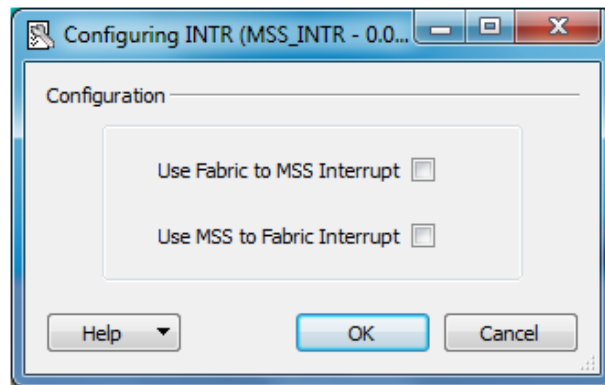


Figure 323 • FIIC Configurator

3. Select either or both check boxes per the application need.
Use Fabric to MSS Interrupt: Use this option to expose the MSS_INT_F2M interrupt port. MSS_INT_F2M signals are then available to be used in the design.
Use MSS to Fabric Interrupt: Use this option to expose MSS_INT_M2F, M3_NMI and COM-M_BLK_INT interrupts ports. These signals are then available to be used in the design.

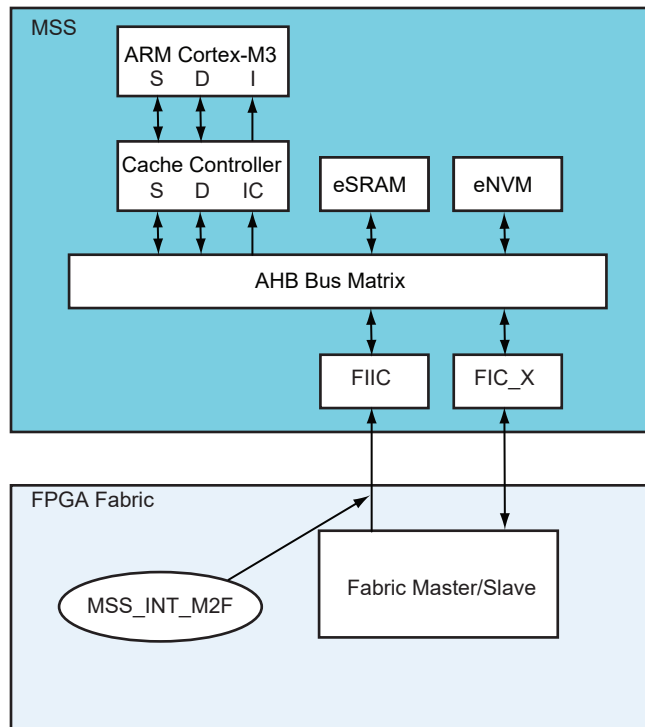
23.3.2 FIIC Use Models

This section explains the use models and gives directions for using the FIIC in an application.

23.3.2.1 Use Model 1: Fabric to MSS Interrupt

The following figure shows fabric master/slave connectivity with the FIIC. Select **Use Fabric to MSS Interrupt** in the Interrupt Management (FIIC) configurator in the Libero SoC. The MSS_INT_F2M signals are then available to be used in the design. The fabric master/slave can be implemented using FSM with APB/AHB-Lite interface. Any user logic (for example, timer/counter) in the FPGA fabric can also be used as a source of interrupt.

Figure 324 • Fabric to the MSS Interrupt



23.3.2.1.1 Software Design Flow

The software design flow consists of enabling the interrupts and the implementation of interrupt handlers. The interrupt handler executes on the occurrence of interrupts. The following is a description of the software application programming interfaces (APIs).

Enabling the Fabric to the MSS Interrupt

This function enables the fabric to the MSS interrupt, MSS_INT_F2M, in the Cortex-M3 NVIC interrupt controller by calling the following API:

```
NVIC_EnableIRQ (FabricIrqX_IRQn);
```

where X can be set from 0 to 15

FabricIrqX_IRQn represents interrupt source numbers, which are connected to the NVIC of the Cortex-M3 processor in the MSS. The following table gives the interrupt source numbers that corresponds to the fabric to MSS interrupt, MSS_INT_F2M, signals.

Table 768 • Interrupt Source Numbers

F2M Interrupt Signal	Interrupt Number	Interrupt Type
MSS_INT_F2M [0]	34	FabricIrq0_IRQn
MSS_INT_F2M [1]	35	FabricIrq1_IRQn
MSS_INT_F2M [2]	36	FabricIrq2_IRQn
MSS_INT_F2M [3]	37	FabricIrq3_IRQn
MSS_INT_F2M [4]	38	FabricIrq4_IRQn
MSS_INT_F2M [5]	39	FabricIrq5_IRQn
MSS_INT_F2M [6]	40	FabricIrq6_IRQn
MSS_INT_F2M [7]	41	FabricIrq7_IRQn
MSS_INT_F2M [8]	42	FabricIrq8_IRQn
MSS_INT_F2M [9]	43	FabricIrq9_IRQn
MSS_INT_F2M [10]	44	FabricIrq10_IRQn
MSS_INT_F2M [11]	45	FabricIrq11_IRQn
MSS_INT_F2M [12]	46	FabricIrq12_IRQn
MSS_INT_F2M [13]	47	FabricIrq13_IRQn
MSS_INT_F2M [14]	48	FabricIrq14_IRQn
MSS_INT_F2M [15]	49	FabricIrq15_IRQn

Refer to the interrupts section in the [Cortex-M3 Processor Overview and Debug Features](#), page 6 for more information.

Fabric to the MSS Interrupt Handler

This interrupt handler executes on the occurrence of the fabric to MSS interrupts. This is done by calling the following API:

```
void FabricIrqX_IRQHandler (void);
```

where X can be set from 0 to 15.

Clearing the Pending Interrupt

The NVIC_ClearPendingIRQ() function is used to clear the interrupt in the Cortex-M3 interrupt controller (NVIC). The following API is used to clear the fabric to the MSS interrupt:

```
NVIC_ClearPendingIRQ (FabricIrqX_IRQn);
```

where X can be set from 0 to 15.

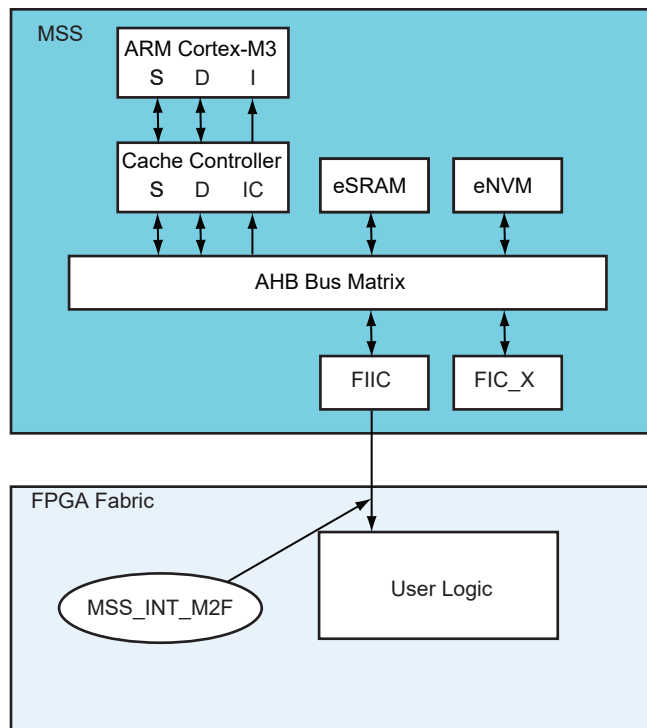
Note: Once Fabric to MSS Interrupt is asserted, user logic in the fabric must keep the interrupt asserted until it is cleared by the Cortex-M3 processor firmware.

23.3.2.2 Use Model 2: MSS to the Fabric Interrupt

The following figure shows user logic in the fabric connectivity with the FIIC. You need to select **Use MSS to Fabric Interrupt** in the Interrupt Management (FIIC) configurator in the Libero SoC. The MSS_INT_M2F signals are then available to be used in the design.

The user logic can monitor these signals and process according to the application requirement. The figure lists the MSS peripheral interrupts that correspond to the MSS_INT_M2F signals.

Figure 325 • MSS to Fabric Interrupt



23.3.2.2.1 Software Design Flow

The software design flow consists of enabling the MSS to the fabric interrupts.

Enabling the MSS to the Fabric interrupt

The interrupt enable registers do not affect the Cortex-M3 process or the NVIC; these are per bit enables of the interrupt routed to the FPGA fabric. It enables the MSS to fabric interrupt, MSS_INT_M2F, by setting the following INTERRUPT_ENABLE0 or INTERRUPT_ENABLE1 register bit-band.

```
INTERRUPT_CTRL_BITBAND-> bit-band register bit of INTERRUPT_ENABLE0 or
INTERRUPT_ENABLE1 = <1/0>;
```

The following table gives the bit-band register bit of INTERRUPT_ENABLE0 and INTERRUPT_ENABLE1 in INTERRUPT_CTRL structure. Refer to m2sxxx.h contained in the CMSIS folder.

Table 769 • Bit-Band Register Bit of Interrupt_Enable0 and Interrupt_Enable1

Bit-Band Register Bit	
INTERRUPT_ENABLE0	INTERRUPT_ENABLE1
SPIINT0_ENBL	RESERVED1[3]
SPIINT1_ENBL	MDDR_IO_CALIB_INT_ENBL
I2C_INT0_ENBL	RESERVED2

Table 769 • Bit-Band Register Bit of Interrupt_Enable0 and Interrupt_Enable1 (continued)

I2C_INT1_ENBL	FAB_PLL_LOCK_INT_ENBL
MMUART0_INTR_ENBL	FAB_PLL_LOCKLOST_INT_ENBL
MMUART1_INTR_ENBL	FIC64_INT_ENBL
MAC_INT_ENBL	RESERVED3[24]
USB_MC_INT_ENBL	
PDMAINTERRUPT_ENBL	
HPD_XFR_CMP_INT_ENBL	
TIMER1_INTR_ENBL	
TIMER2_INTR_ENBL	
CAN_INTR_ENBL	
RTC_WAKEUP_INTR_ENBL	
WDOGWAKEUPINT_ENBL	
MSSDDR_PLL_LOCKLOST_INT_ENBL	
ENVM_INT0_ENBL	
ENVM_INT1_ENBL	
I2C_SMBALERT0_ENBL	
I2C_SMBSUS0_ENBL	
I2C_SMBALERT1_ENBL	
I2C_SMBSUS1_ENBL	
HPD_XFR_ERR_INT_ENBL	
MSSDDR_PLL_LOCK_INT_ENBL	
SW_ERRORINTERRUPT_ENBL	
DDRB_INTR_ENBL	
ECCINTR_ENBL	
CACHE_ERRINTR_ENBL	
SOFTINTERRUPT_ENBL	
COMBLK_INTR_ENBL	
USB_DMA_INT_ENBL	
RESERVED0	

To enable the MSS to the fabric interrupts, listed in the Select Group 0 or Select Group 1 column of [Table 766](#), page 740, set the bit-band register bit of INTERRUPT_MODE register as given below:

```
INTERRUPT_CTRL_BITBAND-> SELECT_MODE= <1/0>;
```

Monitoring the MSS to the Fabric Interrupt Status

The status of the MSS to the fabric interrupt, MSS_INT_M2F, can be monitored by reading the bit-band register bit of INTERRUPT_REASON0 and INTERRUPT_REASON1, as given below:

```
INTERRUPT_CTRL_BITBAND-> bit-band register bit of INTERRUPT_REASON0 or  
INTERRUPT_REASON1;
```

The following table gives the bit-band register bit of INTERRUPT_REASON0 and INTERRUPT_REASON1 in the INTERRUPT_CTRL structure. Refer to m2sxxx.h contained in the CMSIS folder.

Table 770 • Bit-Band Register Bit of INTERRUPT_REASON0 and INTERRUPT_REASON1 .

Bit-Band Register Bit	
INTERRUPT_REASON0	INTERRUPT_REASON1
SPIINT0_STATUS	RESERVED5[3]
SPIINT1_STATUS	MDDR_IO_CALIB_INT_STATUS
I2C_INT0_STATUS	RESERVED6
I2C_INT1_STATUS	FAB_PLL_LOCK_INT_STATUS
MMUART0_INTR_STATUS	FAB_PLL_LOCKLOST_INT_STATUS
MMUART1_INTR_STATUS	FIC64_INT_STATUS
MAC_INT_STATUS	RESERVED7[24]
USB_MC_INT_STATUS	
PDMAINTERRUPT_STATUS	
HPD_XFR_CMP_INT_STATUS	
TIMER1_INTR_STATUS	
TIMER2_INTR_STATUS	
CAN_INTR_STATUS	
RTC_WAKEUP_INTR_STATUS	
WDOGWAKEUPINT_STATUS	
MSSDDR_PLL_LOCKLOST_INT_STATUS	
ENVM_INT0_STATUS	
ENVM_INT1_STATUS	
I2C_SMBALERT0_STATUS	
I2C_SMBSUS0_STATUS	
I2C_SMBALERT1_STATUS	
I2C_SMBSUS1_STATUS	
HPD_XFR_ERR_INT_STATUS	
MSSDDR_PLL_LOCK_INT_STATUS	
SW_ERRORINTERRUPT_STATUS	
DDRB_INTR_STATUS	
ECCINTR_STATUS	
CACHE_ERRINTR_STATUS	
SOFTINTERRUPT_STATUS	
COMBLK_INTR_STATUS	
USB_DMA_INT_STATUS	
RESERVED4	

Example: The following code illustrates the usage of the MSS to the fabric interrupt in conjunction with the Timer1 interrupt.

```
int main()
{
    // STEP 1 - Enable Timer1 MSS to Fabric Interrupt (MSS_INT_M2F[10])
    INTERRUPT_CTRL_BITBAND->TIMER1_INTR_ENBL = 1;

    // STEP 2 - Configure Timer1 in PERIODIC MODE
    MSS_TIM1_init(MSS_TIMER_PERIODIC_MODE );

    // STEP 3 - Load count value
    MSS_TIM1_load_immediate(10000000 );

    // STEP 4 - Start Timer1
    MSS_TIM1_start();

    // STEP 5 - Enable Timer1 Interrupt and IRQ in NVIC
    MSS_TIM1_enable_irq();

    // Foreground loop
    for(;;)
    {
        ;
    }

    return 0;
}

/*
 * Connect MSS_INT_M2F[10] signal to LED.
 * Toggle LED on TIM1 interrupt.
 */
__attribute__((__interrupt__)) void Timer1_IRQHandler( void )
{
    uint32_t timer1_interrupt;

    // Read Timer1 MSS to Fabric Interrupt status
    timer1_interrupt = INTERRUPT_CTRL_BITBAND->TIMER1_INTR_STATUS;

    // Delay for extending the Timer1 MSS to fabric Interrupt pulse width
    delay(10000);

    /* Clear TIM1 interrupt */
    MSS_TIM1_clear_irq();
}
```

23.3.2.2.2 Soft Processor in FPGA fabric

If MSS peripheral interrupt sources to be used as an interrupt sources to a soft processor within the FPGA fabric, the following steps to be implemented in soft processor.

Step 1 - Enabling the MSS to the Fabric interrupt

Set M2F interrupt enable register bit of MSS peripheral in [<INTERRUPT_ENABLE0>](#) or [<INTERRUPT_ENABLE1>](#) registers.

For example, to enable Timer1 MSS to fabric interrupt (MSS_INT_M2F [10]), TIMER1_INTR_ENBL bit in [INTERRUPT_ENABLE0](#) register must be set.

Step 2 - initialize and configure peripheral

Refer to peripheral chapters for initialization, configuration and use model.

Step 3 - Enable peripheral interrupt

Refer to peripheral chapters for interrupt enable registers.

For example, you need to set TIMxINTEN bit in [TIMx_CTRL](#) register for Timer1 interrupt.

Note: Once MSS to Fabric Interrupt is asserted, the interrupt remains asserted until the user logic (soft processor /FSM) clears the appropriate MSS peripheral interrupt clear register.

Steps 4 - Clear peripheral interrupt

Refer to peripheral chapters for interrupt clear registers.

For example, you need to set TIMx_RIS bit in [TIMx_RIS](#) register for clearing Timer1 interrupt.

23.4 FIIC Controller Registers

The register set contains two interrupt enable registers, two interrupt status registers and an interrupt mode register. The interrupt enable registers do not affect the Cortex-M3 processor NVIC; these are per bit enables of the interrupt routed to the FPGA fabric.

The following table summarizes each of the registers covered by this chapter. The base address of the FIIC block is 0x40006000.

Table 771 • SmartFusion2 SoC FPGA FIIC Register Map

Register Name	Address Offset	Register Type	Reset Value	Description
INTERRUPT_ENABLE0	0x00	R/W	0x0	Enables MSS to fabric interrupts
INTERRUPT_ENABLE1	0x04	R/W	0x0	Enables MSS to fabric interrupts
INTERRUPT_REASON0	0x08	RO	0x0	Indicates which interrupts are active
INTERRUPT_REASON1	0x0C	RO	0x0	Indicates which interrupts are active
INTERRUPT_MODE	0x10	R/W	0x0	Indicates select group 0 or select group1

23.5 FIIC Controller Register Bit Definitions

The following tables provide the bit definitions for registers in the FIIC.

Table 772 • INTERRUPT_ENABLE0

Bit Number	Name	Reset Value	Description
0	SPIINT0_ENBL	0	SPIINT0 interrupt from the MSS SPI_0 block to fabric. 1: Enable 0: Mask
1	SPIINT1_ENBL	0	SPIINT1 interrupt from the MSS SPI_1 block to fabric. 1: Enable 0: Mask
2	I2C_INT0_ENBL	0	I2C_INT0 interrupt from the MSS I2C_0 block to fabric. 1: Enable 0: Mask
3	I2C_INT1_ENBL	0	I2C_INT1 interrupt from the MSS I2C_1 block to fabric. 1: Enable 0: Mask

Table 772 • INTERRUPT_ENABLE0 (continued)

Bit Number	Name	Reset Value	Description
4	MMUART0_INTR_ENBL	0	MMUART0_INTR interrupt from the MSS MMUART_0 block to fabric. 1: Enable 0: Mask
5	MMUART1_INTR_ENBL	0	MMUART1_INTR interrupt from the MSS MMUART_1 block to fabric. 1: Enable 0: Mask
6	MAC_INT_ENBL	0	MAC_INT interrupt from the MSS Ethernet MAC block to fabric. 1: Enable 0: Mask
7	USB_MC_INT_ENBL	0	USB_MC_INT interrupt from the MSS USB block to fabric. 1: Enable 0: Mask
8	PDMAINTERRUPT_ENBL	0	PDMAINTERRUPT interrupt from the MSS peripheral DMA block to fabric. 1: Enable 0: Mask
9	HPD_XFR_CMP_INT_ENBL	0	HPD_XFR_CMP_INT interrupt from the MSS HPDMA block to fabric. 1: Enable 0: Mask
10	TIMER1_INTR_ENBL	0	TIMER1_INTR interrupt from the MSS TIMER1 block to fabric. 1: Enable 0: Mask
11	TIMER2_INTR_ENBL	0	TIMER2_INTR interrupt from the MSS TIMER2 block to fabric. 1: Enable 0: Mask
12	CAN_INTR_ENBL	0	CAN_INTR interrupt from the MSS CAN controller block to fabric. 1: Enable 0: Mask
13	RTC_WAKEUP_INTR_ENBL	0	RTC_WAKEUP_INTR interrupt from the MSS RTC block to fabric. 1: Enable 0: Mask
14	WDOGWAKEUPINT_ENBL	0	WDOGWAKEUPINT interrupt from the MSS Watchdog block to fabric. 1: Enable 0: Mask The watchdog is refreshed by writing to the WDOGREFRESH register. When the counter value is greater than the value in the WDOGMVRP register and SLEEPING input to the watchdog is asserted, the WDOGWAKEUPINT interrupt is generated.

Table 772 • INTERRUPT_ENABLE0 (continued)

Bit Number	Name	Reset Value	Description
15	MSSDDR_PLL_LOCKLOST_INT_ENBL	0	MSSDDR_PLL_LOCKLOST_INT interrupt from MPLL to the fabric. 1: Enable 0: Mask
16	ENVM_INT0_ENBL	0	ENVM_INT0 interrupt from the MSS ENVM0 block to fabric. 1: Enable 0: Mask
17	ENVM_INT1_ENBL	0	ENVM_INT1 interrupt from MSS ENVM1 block to fabric. 1: Enable 0: Mask
18	I2C_SMBALERT0_ENBL	0	I2C_SMBALERT0 interrupt from MSS I2C_0 block to fabric. 1: Enable 0: Mask
19	I2C_SMBSUS0_ENBL	0	I2C_SMBSUS0 interrupt from the MSS I2C_0 block to fabric. 1: Enable 0: Mask
20	I2C_SMBALERT1_ENBL	0	I2C_SMBALERT1 interrupt from the MSS I2C_1 block to fabric. 1: Enable 0: Mask
21	I2C_SMBSUS1_ENBL	0	I2C_SMBSUS1 interrupt from the MSS I2C_1 block to fabric. 1: Enable 0: Mask
22	HPD_XFR_ERR_INT_ENBL	0	HPD_XFR_ERR_INT interrupt from the MSS HPDMA block to fabric. 1: Enable 0: Mask
23	MSSDDR_PLL_LOCK_INT_ENBL	0	MSSDDR_PLL_LOCK_INT interrupt from the MPLL block to fabric. 1: Enable 0: Mask

Table 772 • INTERRUPT_ENABLE0 (continued)

Bit Number	Name	Reset Value	Description
24	SW_ERRORINTERRUPT_ENBL	0	<p>SW_ERRORINTERRUPT interrupt from the SYSREG block to fabric.</p> <p>1: Enable 0: Mask</p> <p>HRESP from AHB bus Matrix assertion to the master in case of blocked fabric master or for the unimplemented address space results in SW_ERRORINTERRUPT signal.</p> <p>In case of above error condition the following signals are ORed together in SYSREG to create the SW_ERRORINTERRUPT signal.</p> <ol style="list-style-type: none"> 1. HRESP assertion being issued to the HPDMA. 2. HRESP assertion being issued to FIC_0. 3. HRESP assertion being issued to FIC_1. 4. HRESP assertion being issued to the Ethernet MAC. 5. HRESP assertion being issued to the peripheral DMA engine. 6. HRESP assertion being issued to the USB. 7. HRESP assertion being issued to the system controller.
25	DDRB_INTR_ENBL	0	<p>MSS DDR bridge DDRB_INTR to fabric.</p> <p>1: Enable 0: Mask</p> <p>DDRB_INTR input indicates that any one of the following interrupts are asserted from the MSS DDR bridge:</p> <ul style="list-style-type: none"> DDRB_ERROR interrupts DDRB_DISABLEDONE interrupts DDRB_LOCKTIMEOUT interrupts
26	ECCINTR_ENBL	0	<p>ECCINTR interrupt from ESRAM0, ESRAM1, the cache controller, CAN, MDDR, and USB to fabric.</p> <p>1: Enable 0: Mask. The ECCINTR interrupt is asserted when an SECDED error has been detected in ESRAM0, ESRAM1, the cache controller, CAN, MDDR, or USB memories.</p>
27	CACHE_ERRINTR_ENBL	0	<p>CACHE_ERRINTR interrupt from the cache controller block to fabric.</p> <p>1: Enable 0: Mask</p> <p>The CACHE_ERRINTR interrupt is generated in the SYSREG block by ORing of the following interrupts from the SmartFusion2 SoC FPGA CACHE block:</p> <ul style="list-style-type: none"> CC_HRESPERRINT0 CC_HRESPERRINT1 CC_HRESPERRINT2 CC_HRESPERRINT3 CC_EDCERRINT

Table 772 • INTERRUPT_ENABLE0 (continued)

Bit Number	Name	Reset Value	Description
28	SOFTINTERRUPT_ENBL	0	SOFTINTERRUPT interrupt from the SYSREG block to fabric. 1: Enable 0: Mask SOFTINTERRUPT is set by the Cortex-M3 processor firmware by writing to the soft interrupt SYSREG block bits.
29	COMBLK_INTR_ENBL	0	COMBLK_INTR interrupt from the COMM_BLK block to fabric. 1: Enable 0: Mask
30	USB_DMA_INT_ENBL	0	USB_DMA_INT interrupt from USB's DMA controller to fabric. 1: Enable 0: Mask
31	Reserved	0	Reserved

Table 773 • INTERRUPT_ENABLE1

Bit Number	Name	Reset Value	Description
0	Reserved	0	Reserved
1	Reserved	0	Reserved
2	Reserved	0	Reserved
3	MDDR_IO_CALIB_INT_ENBL	0	MDDR_IO_CALIB_INT interrupt from the MDDR block to fabric. 1: Enable 0: Mask
4	Reserved	0	Reserved
5	FAB_PLL_LOCK_INT_ENBL	0	FAB_PLL_LOCK_INT interrupt from FAB_PLL. 1: Enable 0: Mask
6	FAB_PLL_LOCKLOST_INT_ENBL	0	FAB_PLL_LOCKLOST_INT interrupt from FAB_PLL. 1: Enable 0: Mask
7	FIC64_INT_ENBL	0	FIC64_INT interrupt from the DDR_FIC block. 1: Enable 0: Mask
8-31	Reserved	0	Reserved

Table 774 • INTERRUPT_REASON1

Bit Number	Name	Reset Value	Description
0	Reserved	0	Reserved
1	Reserved	0	Reserved
2	Reserved	0	Reserved

Table 774 • INTERRUPT_REASON1 (continued)

3	MDDR_IO_CALIB_INT_STATUS	0	Set if the interrupt source for MDDR_IO_CALIB_INT is asserted and the MDDR_IO_CALIB_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE1 is High.
4	Reserved	0	Reserved
5	FAB_PLL_LOCK_INT_STATUS	0	Set if the interrupt source for FAB_PLL_LOCK_INT is asserted and the FAB_PLL_LOCK_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE1 is High.
6	FAB_PLL_LOCKLOST_INT_STATUS	0	Set if the interrupt source for FAB_PLL_LOCKLOST_INT is asserted and the FAB_PLL_LOCKLOST_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE1 is High.
7	FIC64_INT_STATUS	0	Set if the interrupt source for FIC64_INT is asserted and the FIC64_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE1 is High.
8-31	Reserved	0	Reserved

Table 775 • INTERRUPT_REASON0

Bit Number	Name	Reset Value	Description
0	SPIINT0_STATUS	0	Set if the interrupt source for SPIINT0 is asserted and the SPIINT0_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
1	SPIINT1_STATUS	0	Set if the interrupt source for SPIINT1 is asserted and the SPIINT1_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
2	I2C_INT0_STATUS	0	Set if the interrupt source for I2C_INT0 is asserted and the I2C_INT0_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
3	I2C_INT1_STATUS	0	Set if the interrupt source for I2C_INT1 is asserted and the I2C_INT1_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
4	MMUART0_INTR_STATUS	0	Set if the interrupt source for MMUART0_INTR is asserted and the MMUART0_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
5	MMUART1_INTR_STATUS	0	Set if the interrupt source for MMUART1_INTR is asserted and the MMUART1_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
6	MAC_INT_STATUS	0	Set if the interrupt source for MAC_INT is asserted and the MAC_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
7	USB_MC_INT_STATUS	0	Set if the interrupt source for USB_MC_INT is asserted and the USB_MC_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
8	PDMAINTERRUPT_STATUS	0	Set if the interrupt source for PDMAINTERRUPT is asserted and the PDMAINTERRUPT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.

Table 775 • INTERRUPT_REASON0 (continued)

Bit Number	Name	Reset Value	Description
9	HPD_XFR_CMP_INT_STATUS	0	Set if the interrupt source for HPD_XFR_CMP_INT is asserted and the HPD_XFR_CMP_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
10	TIMER1_INTR_STATUS	0	Set if the interrupt source for TIMER1_INTR is asserted and the TIMER1_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
11	TIMER2_INTR_STATUS	0	Set if the interrupt source for TIMER2_INTR is asserted and the TIMER2_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
12	CAN_INTR_STATUS	0	Set if the interrupt source for CAN_INTR is asserted and the CAN_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
13	RTC_WAKEUP_INTR_STATUS	0	Set if the interrupt source for RTC_WAKEUP_INTR is asserted and the RTC_WAKEUP_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
14	WDOGWAKEUPINT_STATUS	0	Set if the interrupt source for WDOGWAKEUPINT is asserted and the WDOGWAKEUPINT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
15	MSSDDR_PLL_LOCKLOST_INT_STATUS	0	Set if the interrupt source for MSSDDR_PLL_LOCKLOST_INT is asserted and the MSSDDR_PLL_LOCKLOST_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
16	ENVN_INT0_STATUS	0	Set if the interrupt source for ENVN_INT0 is asserted and the ENVN_INT0_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
17	ENVN_INT1_STATUS	0	Set if the interrupt source for ENVN_INT1 is asserted and the ENVN_INT1_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
18	I2C_SMBALERT0_STATUS	0	Set if the interrupt source for I2C_SMBALERT0 is asserted and the I2C_SMBALERT0_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
19	I2C_SMBSUS0_STATUS	0	Set if the interrupt source for I2C_SMBSUS0 is asserted and the I2C_SMBSUS0_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
20	I2C_SMBALERT1_STATUS	0	Set if the interrupt source for I2C_SMBALERT1 is asserted and the I2C_SMBALERT1_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
21	I2C_SMBSUS1_STATUS	0	Set if the interrupt source for I2C_SMBSUS1 is asserted and the I2C_SMBSUS1_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.

Table 775 • INTERRUPT_REASON0 (continued)

Bit Number	Name	Reset Value	Description
22	HPD_XFR_ERR_INT_STATUS	0	Set if the interrupt source for HPD_XFR_ERR_INT is asserted and the HPD_XFR_ERR_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
23	MSSDDR_PLL_LOCK_INT_STATUS	0	Set if the interrupt source for MSSDDR_PLL_LOCK_INT is asserted and the MSSDDR_PLL_LOCK_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High. MSSDDR_PLL_LOCK_INT interrupt is asserted when MPLL achieves lock.
24	SW_ERRORINTERRUPT_STATUS	0	Set if the interrupt source for SW_ERRORINTERRUPT is asserted and the SW_ERRORINTERRUPT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
25	DDRB_INTR_STATUS	0	Set if the interrupt source for DDRB_INTR is asserted and the DDRB_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
26	ECCINTR_STATUS	0	Set if the interrupt source for ECCINTR is asserted and the ECCINTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
27	CACHE_ERRINTR_STATUS	0	Set if the interrupt source for CACHE_ERRINTR is asserted and the CACHE_ERRINTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
28	SOFTINTERRUPT_STATUS	0	Set if the interrupt source for SOFTINTERRUPT is asserted and the SOFTINTERRUPT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
29	COMBLK_INTR_STATUS	0	Set if the interrupt source for COMBLK_INTR is asserted and the COMBLK_INTR_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
30	USB_DMA_INT_STATUS	0	Set if the interrupt source for USB_DMA_INT is asserted and USB_DMA_INT_ENBL interrupt enable bit in INTERRUPT_ENABLE0 is High.
31	Reserved	0	Reserved

Table 776 • INTERRUPT_MODE

Bit Number	Name	Reset Value	Description
0	SELECT_MODE	0	The following are the valid values for this bit: 0: Select group 0 1: Select group 1
31:1	Reserved	0	Reserved

24 Fabric Interface Controller

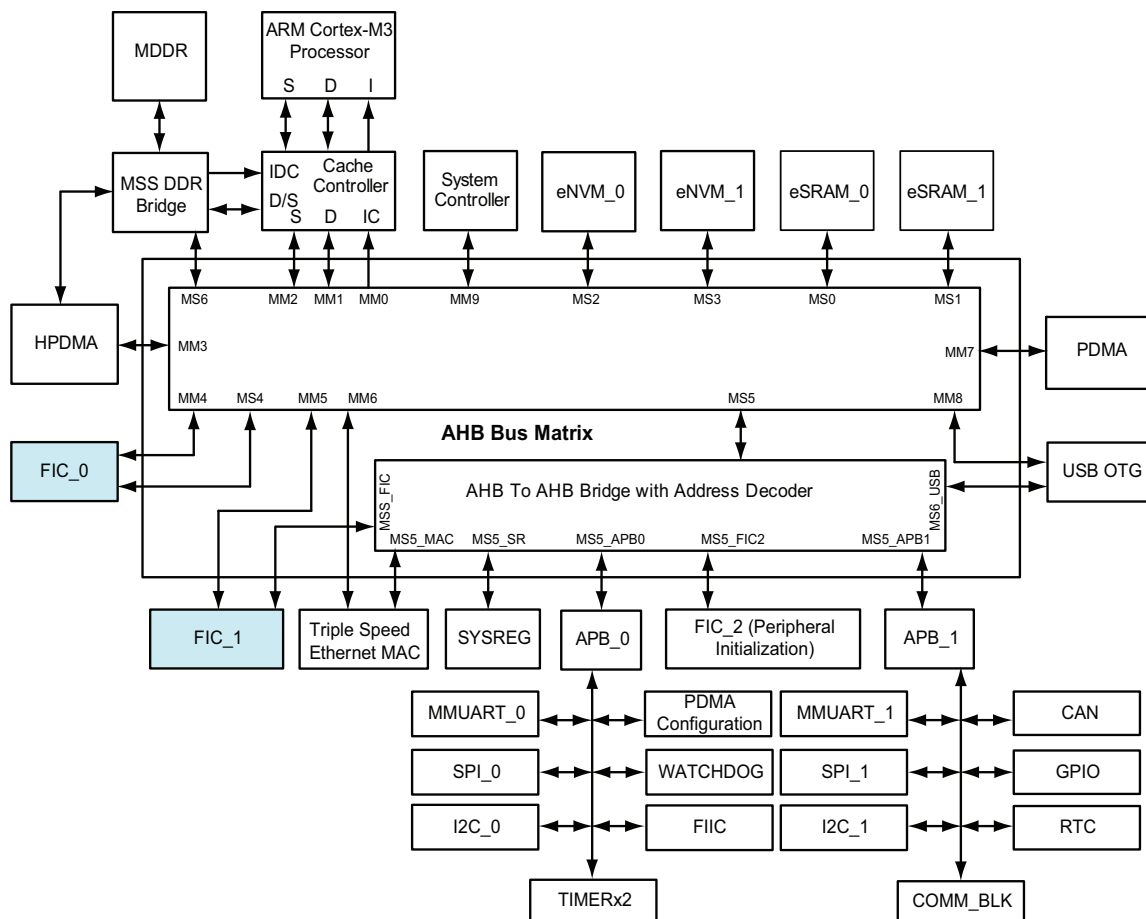
The fabric interface controller (FIC) enables connectivity between the fabric and microcontroller subsystem (MSS). The FIC is a part of the MSS and performs a bridging function for AHB-Lite to APB or AHB-Lite to AHB-Lite between the AHB bus matrix and the FPGA fabric. The interface type is configurable. There are up to two, 32-bit FICs in SmartFusion2 devices, referred to as FIC_0 and FIC_1. Both FICs provide two bus interfaces between the MSS and the fabric. The first is mastered by the MSS and has slaves in the fabric; the second is mastered by the fabric and has slaves in the MSS.

The interfaces to the fabric can be 32-bit AHB-Lite or 32-bit APB. The address and data buses between the FIC and the FPGA fabric are overlaid; hence, only one type of interface can be enabled at any time. However, separate groups of signals are used for the AHB-Lite and APB control signals. In addition to the choice of AHB-Lite or APB interfaces between the MSS and the fabric, a number of options related to relative clock frequencies and pipelining of transactions are available. Each FIC block can operate on a different clock frequency, defined as a ratio of the MSS main clock, M3_CLK.

The SmartFusion2 architecture imposes a certain number of rules related to clocking domains between the fabric interfaces and the FPGA fabric. This document provides guidance on how to properly construct such systems. The following figure depicts the connectivity of FIC_0 and FIC_1 to the AHB bus matrix.

AHB-Lite master interface and AHB-Lite slave interface of FIC_0 is directly connected with mirrored master 4 (MM4) and mirrored slave 4 (MS4) on the AHB bus matrix. To reduce the load on the AHB bus matrix, FIC_1 AHB-Lite slave interface is connected through the synchronous AHB-to-AHB bridge with an address decoder which inserts a one-cycle delay in each direction.

Figure 326 • The FIC Connection to the AHB Bus Matrix



The following table lists the number of FICs available for use in each device.

Table 777 • Number of FICs Available for Use in Each Device

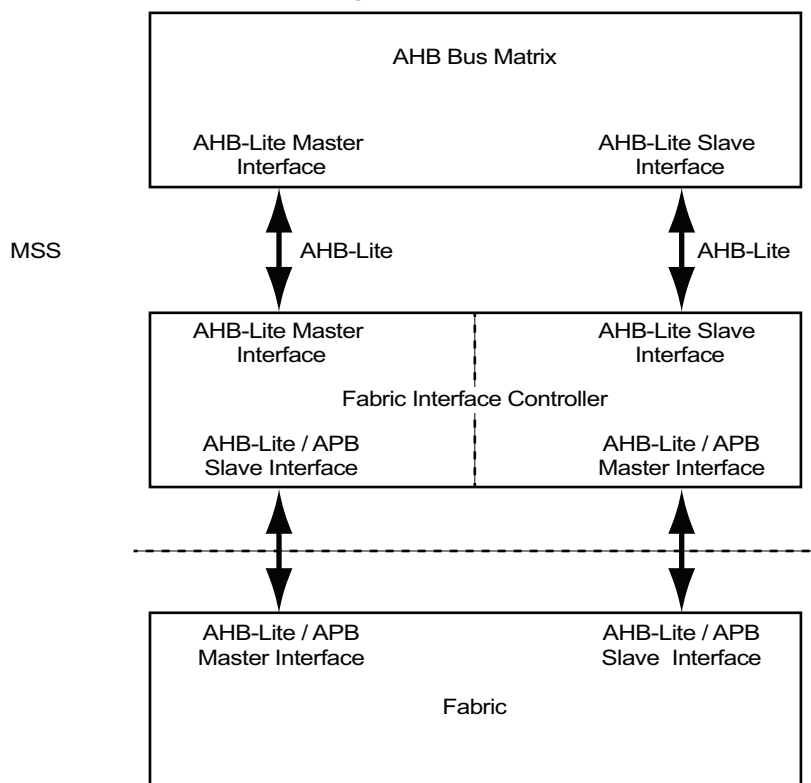
Device	FIC Blocks
M2S005	1 ¹
M2S010	
M2S025	
M2S050	2
M2S060	1*
M2S090	
M2S150	2

1. Only FIC_0 is available.

24.1 Functional Description

This following sections provide a detailed description of the FIC subsystem.

Figure 327 • Fabric Interface Controller Block Diagram



The preceding figure shows a block diagram for the FIC. The FIC is a hard block; enabling or disabling it will not consume any user logic. You can configure FIC_0 and FIC_1 independently from the Libero SoC MSS configurator. The following configuration options are available.

- The MSS to the FPGA fabric interface
- Advanced AHB-Lite options
- The FPGA fabric Address Regions (MSS Master View)

24.1.1 MSS to the FPGA Fabric Interface

The FIC interface can be configured towards the fabric to support AHB-Lite or APB. FIC configuration allows you to implement AHB-Lite or APB slave user logic in the fabric that can expose the memory map of the Cortex-M3 processor and other masters on the AHB bus matrix. You can also implement an AHB-Lite or APB master in the fabric that can access any slave on the AHB bus matrix. Since FIC_0 and FIC_1 have an AMBA interface towards the fabric, user logic should implement the AMBA AHB-Lite or APB3 protocol in order to communicate with the FIC.

The following options are available for implementing peripherals in the fabric.

- If you are using a mix of AHB-Lite and APB peripherals, use CoreAHBLite, CoreAHB2APB3, and CoreAPB3 soft IPs.
- If you are using APB (APB v3.0) peripherals (for example, CoreUARTAPB), use CoreAPB3 soft IP for connecting to the fabric.
- If you are using AHB-Lite peripherals, use CoreAHBLite soft IP for connecting to the fabric interface.

24.1.2 Configure FIC for Master or Slave Interface

FIC_0 and FIC_1 are configured individually through the Libero SoC MSS configurator. There are two options.

- The MSS is the master and the fabric has the slave (HM – hard master).
- The fabric has the master and the MSS is the slave (FM – fabric master).

The MSS side of the FIC has two (master and slave) AHB-Lite interfaces to the AHB bus matrix, as shown in [Figure 327](#), page 758. On the fabric side of the FIC, the master and slave interfaces can be AHB-Lite or APB. Both interfaces on the fabric side always use the same protocol, either AHB-Lite or APB. However, it is possible to have master and slave at the same time.

24.2 Advanced AHB-Lite Options

24.2.1 Configure FIC in Bypass Mode or Synchronous Pipelined Mode

You can configure FIC_0 and FIC_1 individually through the Libero SoC MSS configurator. The AHB-Lite configuration in the FIC configurator provides the Use Bypass Mode option to enable or disable the address and data pipelining between FPGA fabric logic and the AHB bus matrix. In some scenarios, the FPGA fabric logic needs to access the MSS peripherals (such as eSRAM or eNVM) with very high throughput. In such cases, the FPGA fabric logic should be connected to the FIC using an AHB-Lite interface.

In bypass mode (non-pipelined mode / Use Bypass Mode option checked), it is possible to achieve zero-wait state access between the FPGA master and a zero-wait state capable MSS slave, if there is no other master accessing that slave. However, the setup time requirement of the FIC interface is a bigger, which may lower overall frequency of operation. The clock ratio between M3_CLK, FIC_0_CLK, and FIC_1_CLK, must be set to 1:1 when bypass mode is selected. This requirement is enforced in the MSS CCC configurator when bypass mode is selected.

In Pipelined mode (Use Bypass Mode option unchecked / default mode), the interface between the AHB bus matrix and FPGA has registered signals that reduce setup requirements. This may improve overall system frequency, but these registers introduce a bubble in the AHB transaction pipe. It results in inserting a wait-state for each transaction even if the Master and Slave are capable of zero-wait state access. Relative clock frequency between the MSS clock, M3_CLK, and the fabric clock for Synchronous Pipelined mode can be 1:1, 2:1, 4:1, 8:1, 16:1, or 32:1.

You have to analyze the critical paths between the FIC and the logic in the FPGA fabric, when Use Bypass Mode is enabled. You need to make sure that all the timing requirements have been met between FIC and FPGA fabric logic.

FIC32_0_DIVISOR[2:0] and FIC32_1_DIVISOR[2:0] configuration inputs from the SYSREG block MSSDDR_FACC1_CR configuration register, specify the ratio of clocks between the MSS system clock, M3_CLK, and the fabric clock used by the soft IP interfacing with FIC_0 and FIC_1. The FAB0_AHB_BYPASS and FAB1_AHB_BYPASS fields from the SYSREG block FAB_IF_CR register, configure FIC_0 and FIC_1 in Bypass mode or Synchronous Pipelined mode.

24.2.2 Master Identity Port to the Fabric

The AHB bus matrix provides a 2-bit side band signal to the FPGA fabric (one 2-bit signal per FIC instance). The side band signal indicates to the slave, which is implemented in the FPGA fabric, the identification of the master performing the current transaction. These signals have the same timing as other AHB Lite master signals such as HTRANS, HMASTLOCK etc. [Table 778 on page 760](#) provides the decoding of the master accessing the FPGA fabric slave through the MSS AHB bus matrix.

The Libero SoC MSS configurator allows exposure of the master ID port; if the interface is selected to act as a master of the FPGA fabric.

Table 778 • Master Group Access to Fabric Slaves

FIC_X_MASTER_ID	Accessing Master
00	IC-bus, D-bus, and SBus master
01	FIC_0, FIC_1
10	HPDMA, Ethernet MAC, PDMA, USB
11	System controller

24.2.3 Configure MSS Master View for the FPGA Fabric Address

There are six 256 MB regions defined as FIC Regions 0 to 5 in the MSS memory map. Each of these regions can be allocated to the FIC_0 or FIC_1 slave interfaces in a mutually exclusive fashion. Libero SoC MSS configurator allows you to configure the Memory Regions for the FIC Interfaces. By default, fabric regions 0, 1, and 2 are accessible through FIC_0 and regions 3, 4, and 5 are accessible through FIC_1. [Table 779, page 760](#) lists the FIC memory regions.

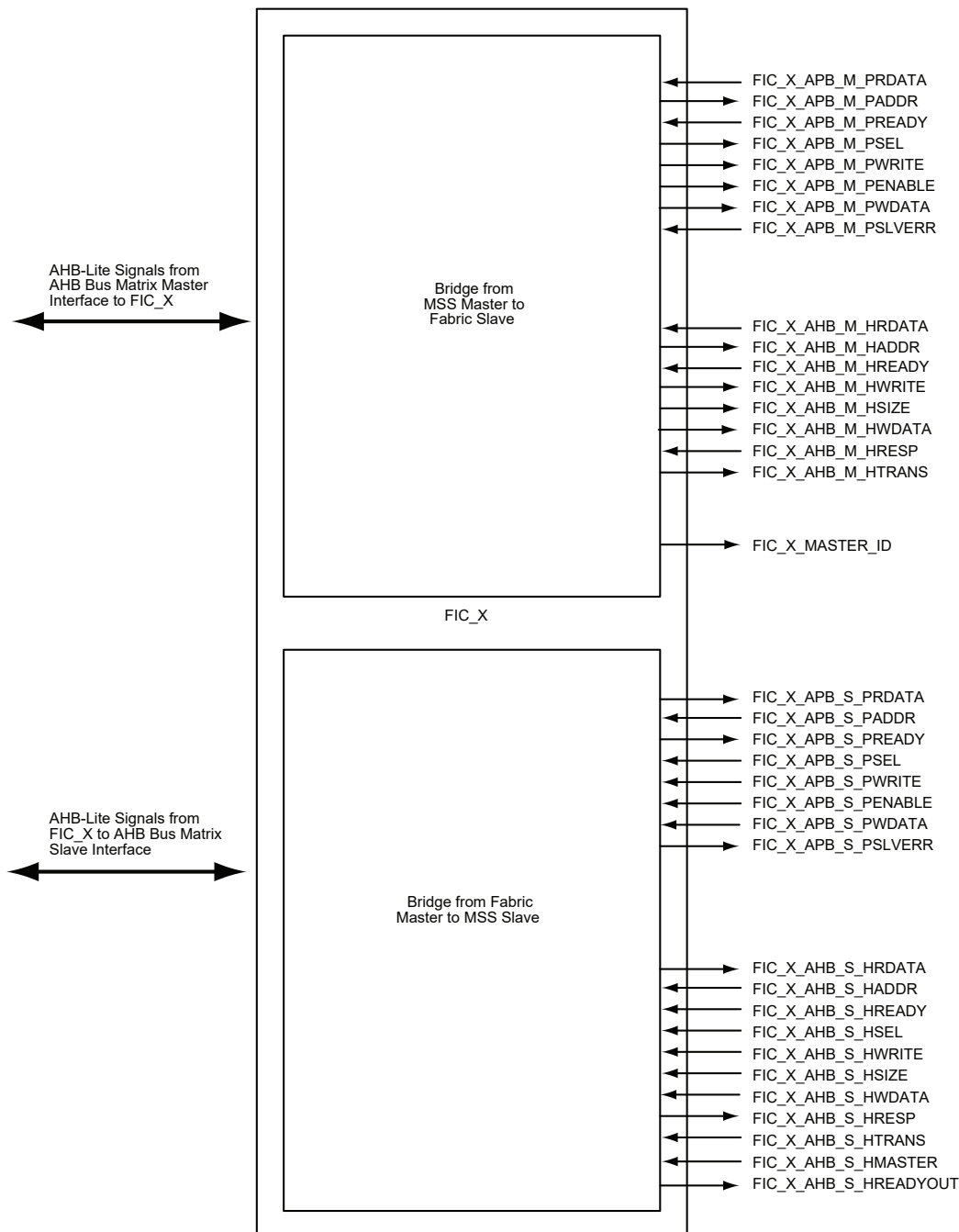
Table 779 • FIC Memory Regions

FIC Region	Start Address	End Address
0	0x30000000	0x3FFFFFFF
1	0x50000000	0x5FFFFFFF
2	0x70000000	0x7FFFFFFF
3	0x80000000	0x8FFFFFFF
4	0x90000000	0x9FFFFFFF
5	0xF0000000	0xFFFFFFFF

24.3 FIC Interface Port List

There are two interfaces between the microcontroller subsystem (MSS) and the fabric. One interface allows an AHB-Lite master in the MSS to communicate with AHB-Lite or APB slaves in the fabric. The second interface allows AHB-Lite or APB masters, that are implemented in the fabric, to communicate with AHB-Lite slaves in the MSS, as shown in the following figure. The bridge from the MSS master to the fabric slave, implements AHB-Lite to AHB-Lite or APB translation; and the bridge from the fabric master to the MSS slave implements AHB-Lite or APB to AHB-Lite translation.

Figure 328 • Fabric Interface Controller Top-Level View



The following table contains the FIC port list.

Table 780 • Fabric Interface Controller Port List

Port Name	Direction	Description
FIC_X_MASTER_ID [1:0]	Out	Indicates the current master performing the transfer. Refer to Table 778 on page 760 .
FIC_X_APB_S_PRDATA [31:0]	Out	Indicates APB read data to the fabric master.
FIC_X_APB_S_PADDR [31:0]	In	Indicates APB address initiated by the fabric master.
FIC_X_APB_S_PREADY	Out	Indicates APB ready signal to the fabric master.
FIC_X_APB_S_PSEL	In	Indicates APB slave select signal from the fabric master.
FIC_X_APB_S_PWRITE	In	Indicates APB write control signal from the fabric master.
FIC_X_APB_S_PENABLE	In	Indicates APB enable from the fabric master. The enable signal is used to indicate the second cycle of an APB transfer.
FIC_X_APB_S_PWDATA [31:0]	In	Indicates APB write data from the fabric master.
FIC_X_APB_S_PSLVERR	Out	Indicates error condition on an APB transfer to the fabric master.
FIC_X_APB_M_PRDATA [31:0]	In	Indicates APB read data from the fabric slave.
FIC_X_APB_M_PADDR [31:0]	Out	Indicates APB address to the fabric slave.
FIC_X_APB_M_PREADY	In	Indicates APB ready signal from the fabric slave.
FIC_X_APB_M_PSEL	Out	Indicates APB slave select signal to the fabric slaves.
FIC_X_APB_M_PWRITE	Out	Indicates APB write control signal to the fabric slaves.
FIC_X_APB_M_PENABLE	Out	Indicates APB enable to the fabric slave. The enable signal is used to indicate the second cycle of an APB transfer.
FIC_X_APB_M_PWDATA [31:0]	Out	Indicates APB write data to the fabric slave.
FIC_X_APB_M_PSLVERR	In	Indicates error condition on an APB transfer from the fabric slave.
FIC_X_AHB_S_HRDATA [31:0]	Out	Indicates AHB read data to the fabric master.
FIC_X_AHB_S_HADDR [31:0]	In	Indicates AHB address initiated by the fabric master.
FIC_X_AHB_S_HREADY	In	Indicates that a transfer has completed on the bus. The fabric master can drive this signal Low to extend a transfer.
FIC_X_AHB_S_HWDATA [31:0]	In	Indicates AHB write data from the fabric master.
FIC_X_AHB_S_HWRITE	In	Indicates AHB write control signal from the fabric master.
FIC_X_AHB_S_HRESP	Out	Indicates AHB transfer response to the fabric master.
FIC_X_AHB_S_HSIZE [1:0]	In	Indicates AHB transfer size from the fabric master.
FIC_X_AHB_S_HTRANS [1:0]	In	Indicates AHB transfer type from the fabric master.
FIC_X_AHB_S_HMASTLOCK	In	Indicates AHB master lock signal from the fabric master.
FIC_X_AHB_S_HSEL	In	Indicates AHB slave select signal from the fabric master.
FIC_X_AHB_S_HREADYOUT	Out	Indicates that a transfer has completed on the bus. The signal is asserted Low to extend a transfer. Input to the fabric master.
FIC_X_AHB_M_HWRITE	Out	Indicates AHB write control signal to the fabric slave.
FIC_X_AHB_M_HADDR [31:0]	Out	Indicates AHB address to the fabric slave.

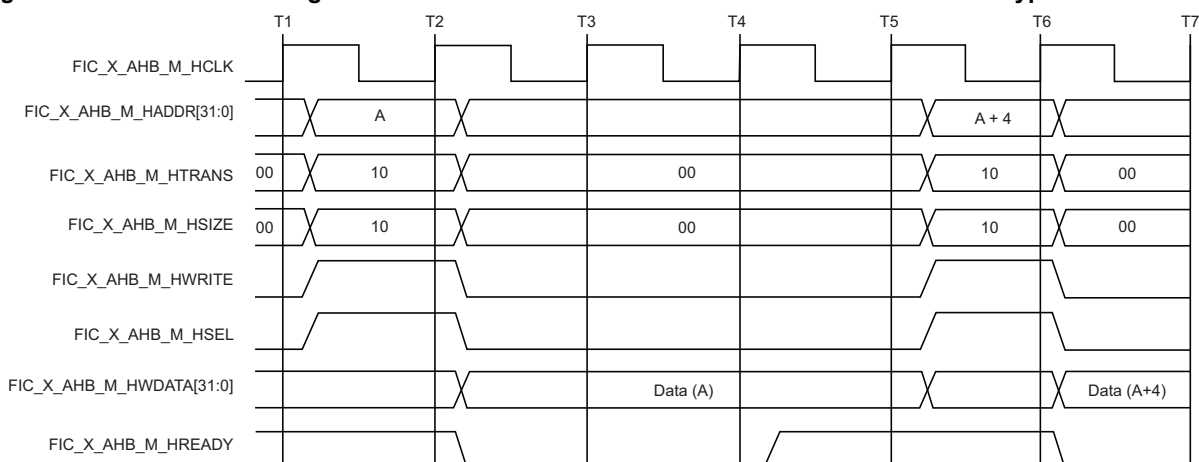
Table 780 • Fabric Interface Controller Port List (continued)

Port Name	Direction	Description
FIC_X_AHB_M_HREADY	In	Indicates that a transfer has completed on the bus. The fabric slave can drive this signal Low to extend a transfer.
FIC_X_AHB_M_HWDATA [31:0]	Out	Indicates AHB-Lite write data to the fabric slave.
FIC_X_AHB_M_HRDATA [31:0]	In	Indicates AHB-Lite read data from the fabric slave.
FIC_X_AHB_M_HRESP	In	Indicates AHB-Lite transfer response from the fabric slave.
FIC_X_AHB_M_HSIZE [1:0]	Out	Indicates AHB-Lite transfer size to the fabric slave.
FIC_X_AHB_M_HTRANS [1:0]	Out	Indicates AHB-Lite transfer type to the fabric slave.

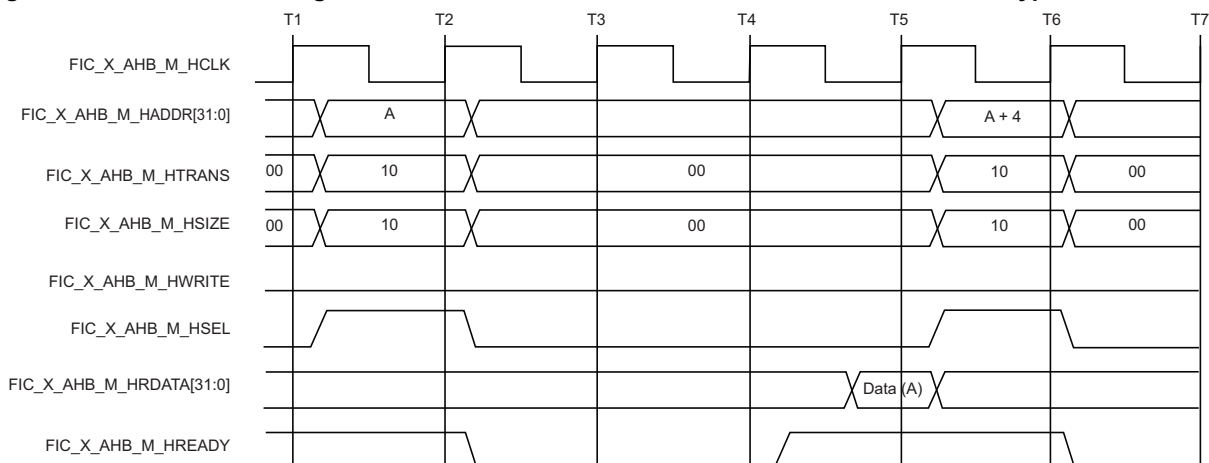
24.4 Timing Diagrams

The timing diagrams contained in this section show AHB-Lite non-sequential transfers with 32 bits as the transfer size.

The following diagram shows the AHB-Lite bus signals from fabric interface controller to the fabric slave for a write transaction in Bypass mode.

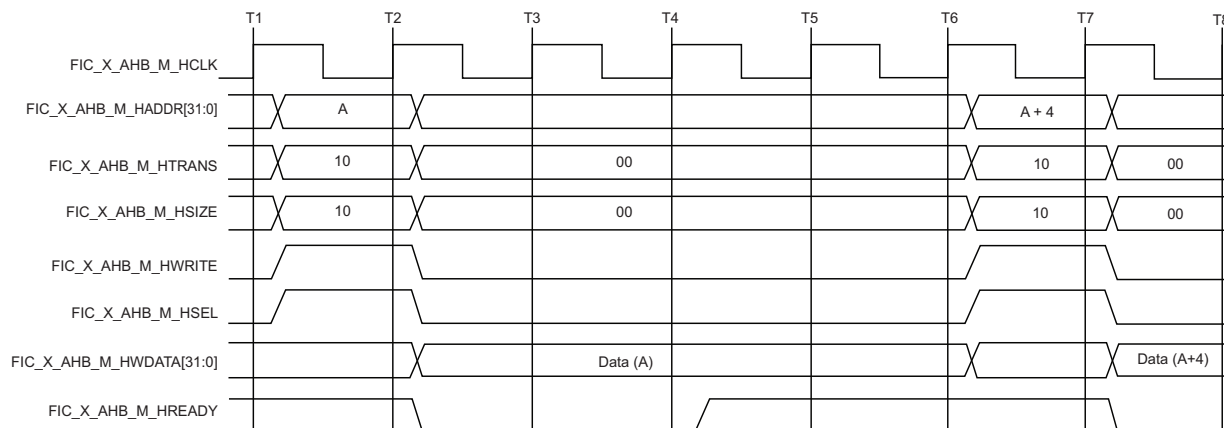
Figure 329 • AHB-Lite Bus Signals from FIC to the Fabric Slave for a Write Transaction in Bypass Mode

The following diagram shows the AHB-Lite bus signals from the fabric interface controller to the fabric slave for a read transaction in Bypass mode.

Figure 330 • AHB-Lite Bus Signals from FIC to the Fabric Slave for a Read Transaction in Bypass Mode

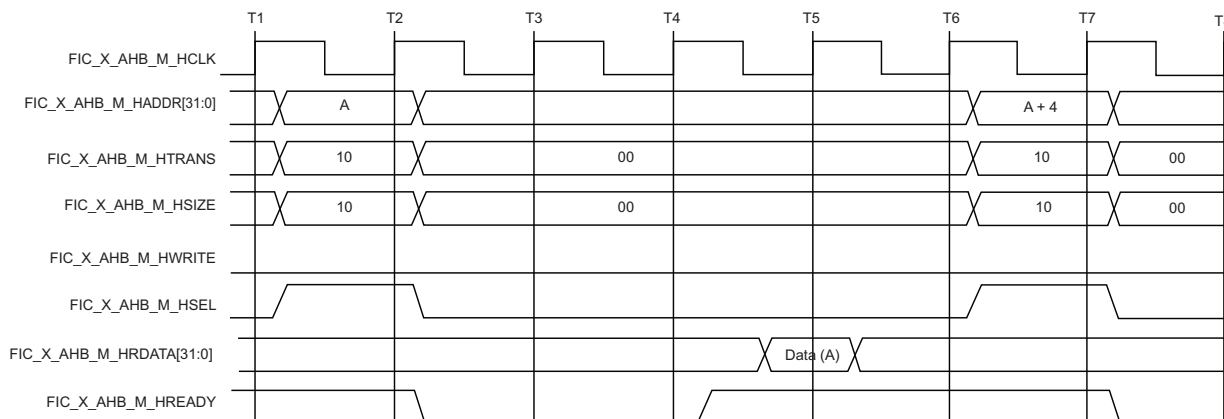
The following diagram shows the AHB-Lite bus signals from the fabric interface controller to the fabric slave for write transaction in Synchronous Pipelined mode.

Figure 331 • AHB-Lite Bus Signals from FIC to the Fabric Slave for a Write Transaction in Synchronous Pipelined Mode



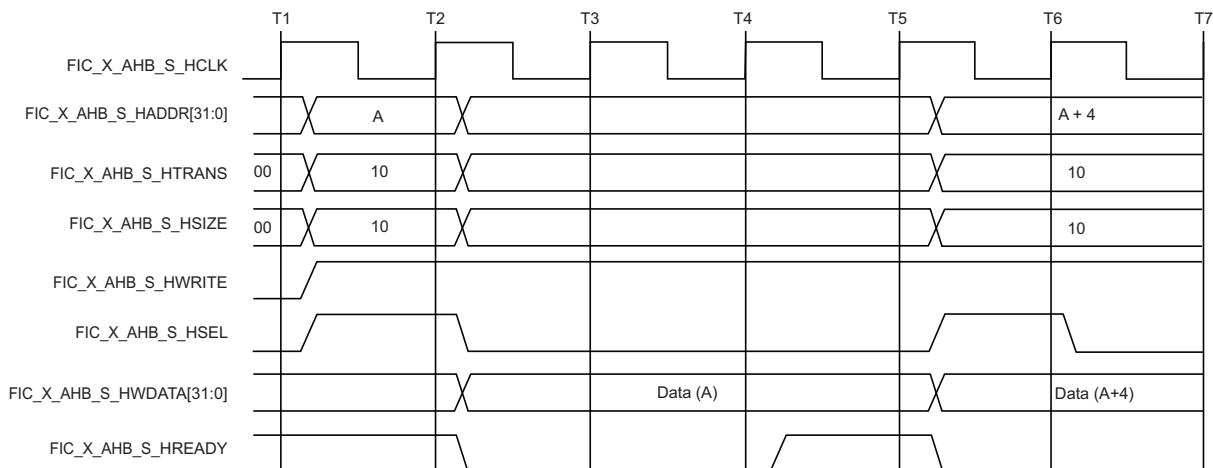
The following diagram shows the AHB-Lite bus signals from the fabric interface controller to the fabric slave for read transaction in Synchronous Pipelined mode.

Figure 332 • AHB-Lite Bus Signals from FIC to the Fabric Slave for a Read Transaction in Synchronous Pipelined Mode



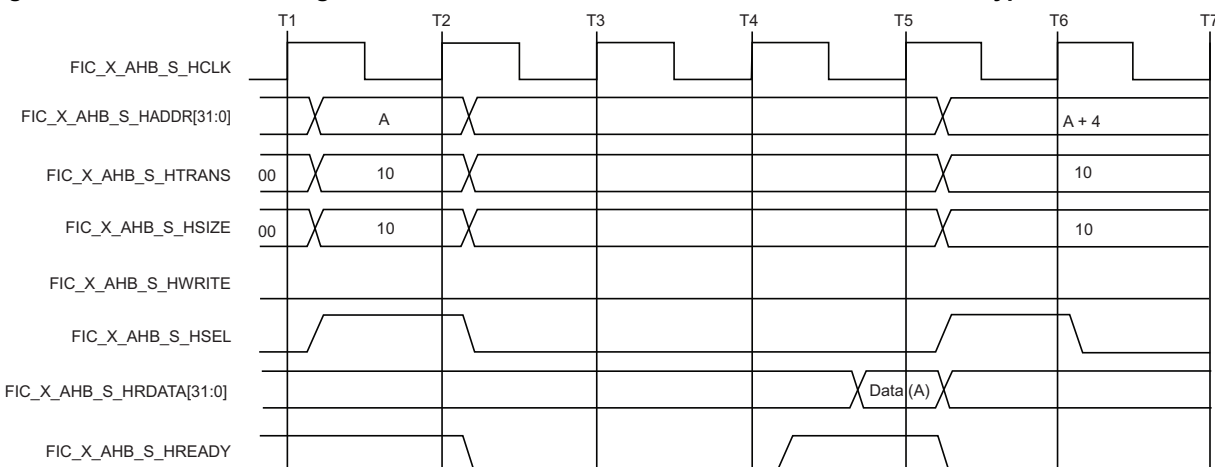
The following diagram shows the AHB-Lite bus signals from the fabric master to the fabric interface controller for write transactions in Bypass mode. Generation of pipelined requests depends on the efficiency of the master in the fabric to generate it.

Figure 333 • AHB-Lite Bus Signals from Fabric Master to FIC for a Write Transaction in Bypass Mode



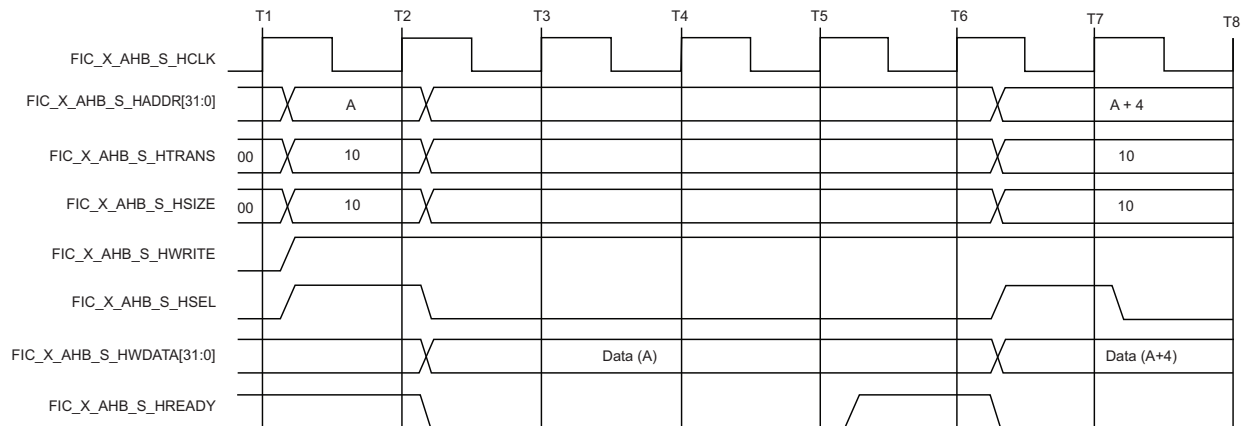
The following diagram shows the AHB-Lite bus signals from the fabric master to the fabric interface controller for read transactions in Bypass mode. Generation of pipelined requests depends on the efficiency of the master in the fabric to generate it.

Figure 334 • AHB-Lite Bus Signals from Fabric Master to FIC for a Read Transaction in Bypass Mode



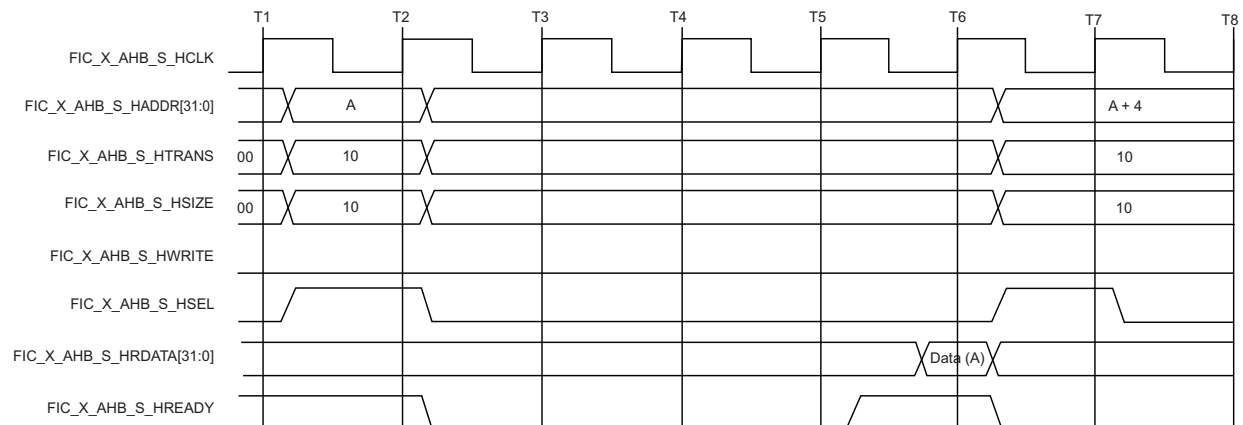
The following diagram shows the AHB-Lite bus signals from the fabric master to the fabric interface controller for write transactions in Synchronous Pipelined mode. Generation of pipelined requests depends on the efficiency of the master in the fabric to generate it.

Figure 335 • AHB-Lite Bus Signals from Fabric Master to FIC for a Write Transaction in Synchronous Pipelined Mode



The following diagram shows the AHB-Lite bus signals from the fabric master to the fabric interface controller for read transactions in Synchronous Pipelined mode. Generation of pipelined requests depends on the efficiency of the master in the fabric to generate it.

Figure 336 • AHB-Lite Bus Signals from Fabric Master to FIC for a Read Transaction in Synchronous Pipelined Mode



Note: When the Fabric master accesses MSS slave through the FIC_1 AHB-Lite slave interface, the AHB-to-AHB bridge inserts a one-cycle delay in each direction. Since these timing diagrams are at FIC interface level, the delay cannot be noticed.

24.5 Implementation Considerations

In AHB mode, the user may perform byte, half word and word accesses from the fabric to MSS. However, in APB16 mode, the user can only cause a word access to occur to an MSS slave. This is done by two accesses over the APB16, one of which is to write a 16-bit holding register (in the case of writes) or to read a 16-bit holding register in the case of reads.

24.6 Fabric Interface Clocks

The fabric alignment clock controller (FACC) block in the MSS DDR clock controller is responsible for the alignment of fabric related clocks. The FACC is interfaced with MSS PLLs (MPLLs) in order to generate the various aligned clocks required by the MSS peripherals and the DDR controller in the MSS (MDDR). The lowest frequency clock, of the aligned clocks being used within the fabric, is fed to the MSS DDR clock controller and is referred to as CLK_BASE. CLK_BASE is internally multiplied and divided within

the ASIC blocks in the MSS to generate higher frequency clocks that are aligned with CLK_BASE; the positive edges of CLK_BASE and derived clocks occur at the same time.

Refer to the [UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide](#) for more details on the alignment of fabric clocks and derived clocks in the MSS.

24.7 How to Use FIC

This section describes how to use the FIC subsystem in the design.

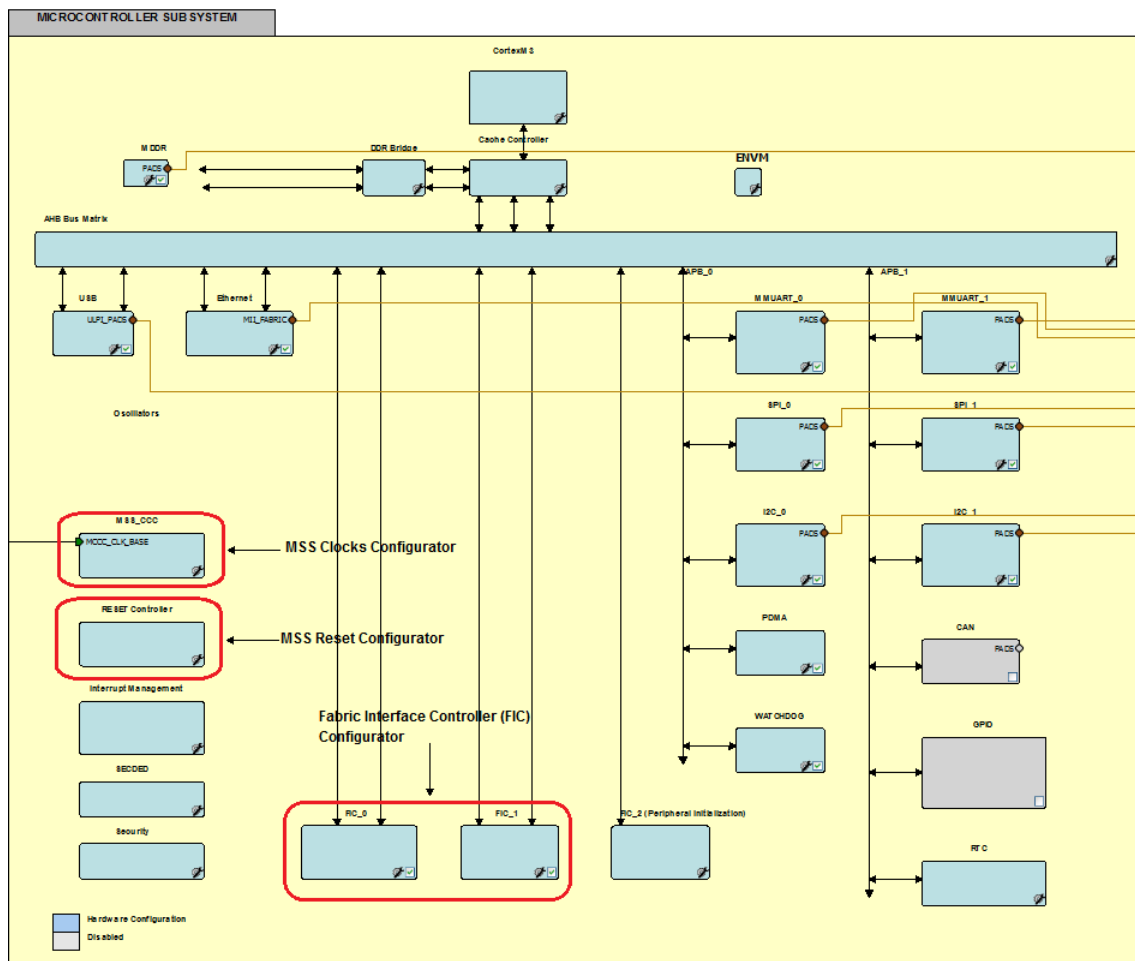
24.7.1 FIC Configuration

The FIC_0 and FIC_1 are not configured by default in the MSS configurator, when the Libero SoC project is created. To configure/create a FIC subsystem:

1. The MSS FIC has to be configured to expose the FIC interface.
2. The FPGA fabric FIC subsystem has to be created including instantiation / configuration / connectivity for:
 - APB or AHB-Lite bus.
 - APB and AHB-Lite compliant master and/or peripherals configuration and connection onto the bus, as required by your application.
 - Clocks and resets; refer to [Configuring the FIC Subsystem Clocks](#), page 774 and [Configuring the FIC Subsystem Reset](#), page 778.

These steps are described in detail below. FIC, Clocks, and Reset sub-blocks are outlined in red in [Figure 328](#), page 761.

Figure 337 • MSS Configurator



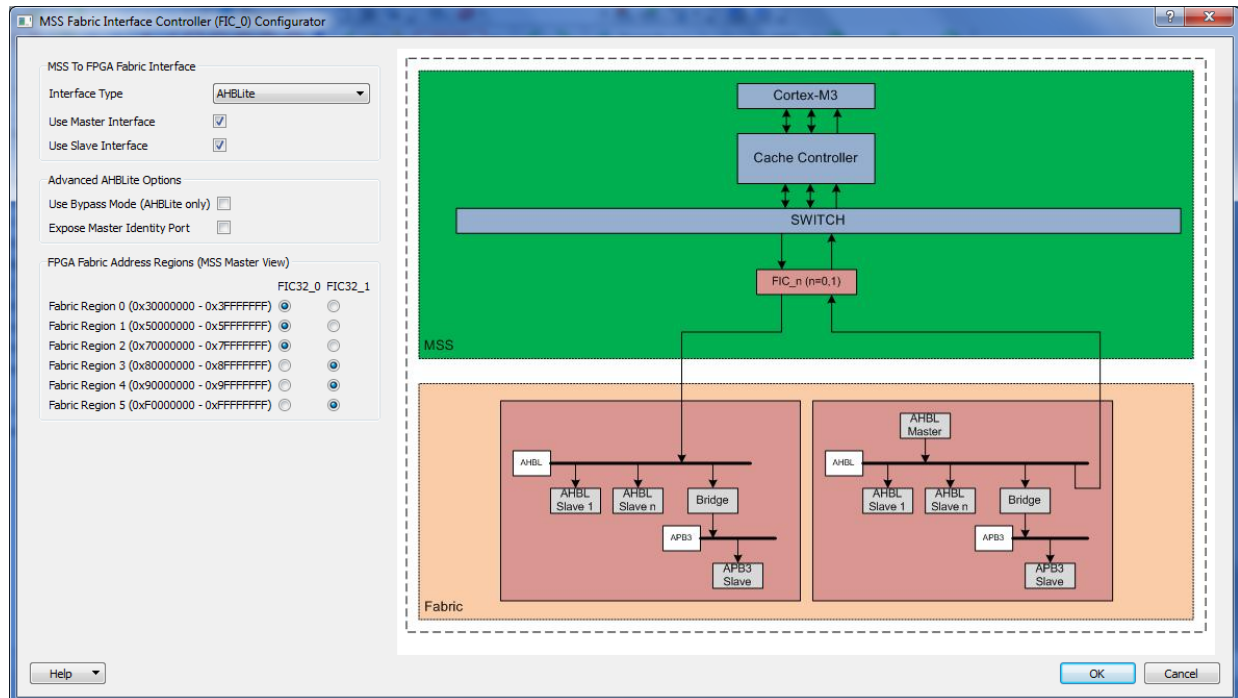
24.7.1.1 Step 1: Configure the MSS FIC Sub-Block

As shown in the following figure, the FIC configurator (applies to both FIC_0 and FIC_1) is organized as follows. In the left panel, the following can be configured:

- The MSS to the FPGA fabric interface
- Advanced AHB-Lite options
- The FPGA fabric address regions (MSS master view) – available in FIC_0 configurator only

In the right panel, a dynamic picture displays the high level block diagram of the architecture chosen. The picture changes when any option in the MSS To FPGA fabric interface group is configured.

Figure 338 • FIC Configurator



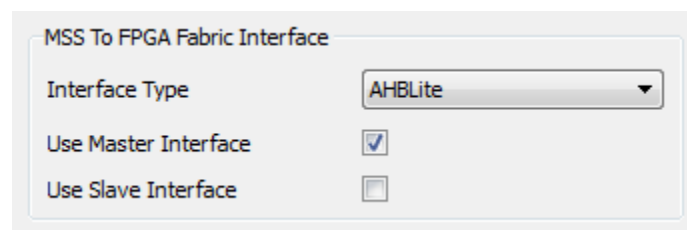
24.7.1.1.1 MSS to the FPGA Fabric Interface

Interface Type: Use this option to select between the AMBA APB (AHB to APB bridge) and AHB-Lite (AHB to AHB bridge) FIC modes (as shown in the following figure).

Use Master Interface: Use this option to expose the Master Bus Interface (BIF) port. When selected, the port is automatically available on the MSS core.

Use Slave Interface: Use this option to expose the Slave Bus Interface (BIF) port. When selected, the port is automatically available on the MSS core.

Figure 339 • MSS to FPGA Fabric Interface Core

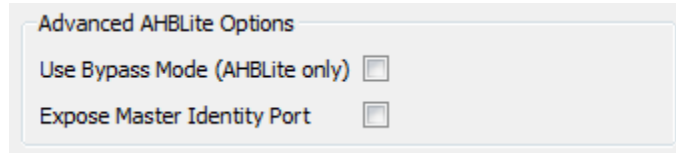


24.7.1.1.2 Advanced AHB-Lite Options

Use Bypass Mode: Use this option to enable the FIC Bypass mode. This option is only active when the interface type is AHB-Lite (as shown in the following figure). The clock ratio between M3_CLK,

FIC_0_CLK, and FIC_1_CLK must be set to 1:1 when bypass mode is selected. This requirement is enforced in the MSS CCC Configurator when bypass is selected.

Figure 340 • Advanced Options Configuration



Advanced AHBLite Options

Use Bypass Mode (AHBLite only) ☐

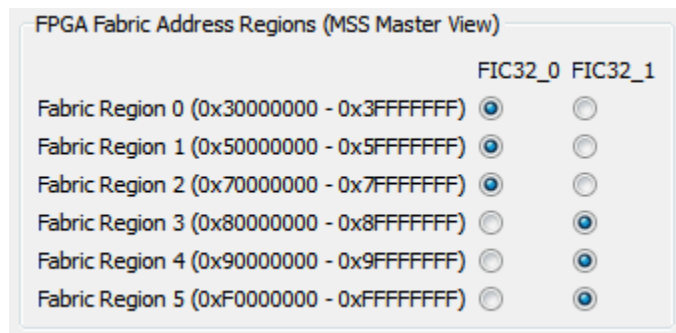
Expose Master Identity Port ☐

Expose Master Identity Port: Use this option to expose a 2-bit side band signal to the FPGA fabric (one 2-bit signal per FIC instance). This option is only active if you selected the interface to act as a master of the fabric.

24.7.1.1.3 FPGA Fabric Address Regions (MSS Master View)

Up to six different memory regions can be assigned to each FIC in the MSS memory map. By default, fabric regions 0, 1, and 2 are accessible through FIC_0, and regions 3, 4, and 5 are accessible through FIC_1 as shown in the following figure.

Figure 341 • FPGA Fabric Address Regions (MSS Master View)



FPGA Fabric Address Regions (MSS Master View)

	FIC32_0	FIC32_1
Fabric Region 0 (0x30000000 - 0x3FFFFFFF)	<input checked="" type="radio"/>	<input type="radio"/>
Fabric Region 1 (0x50000000 - 0x5FFFFFFF)	<input checked="" type="radio"/>	<input type="radio"/>
Fabric Region 2 (0x70000000 - 0x7FFFFFFF)	<input checked="" type="radio"/>	<input type="radio"/>
Fabric Region 3 (0x80000000 - 0x8FFFFFFF)	<input type="radio"/>	<input checked="" type="radio"/>
Fabric Region 4 (0x90000000 - 0x9FFFFFFF)	<input type="radio"/>	<input checked="" type="radio"/>
Fabric Region 5 (0xF0000000 - 0xFFFFFFFF)	<input type="radio"/>	<input checked="" type="radio"/>

Note: This option is available in FIC_0 configurator only. If memory regions are required to be configured to FIC_1, the FIC_0 configurator needs to be opened.

24.7.1.2 Step 2: Create the FPGA Fabric FIC Subsystem

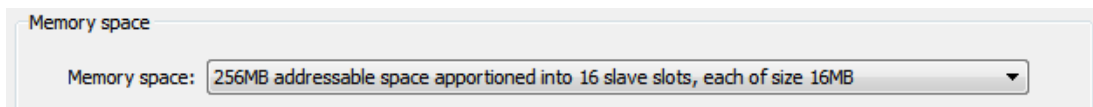
For each FIC interface—master and slave—exposed, a bus (CoreAHBLite or CoreAPB3) must be instantiated that matches the type selected. Depending on the interface role (master/slave) and type (AHB-Lite/APB), the bus configuration is described in the following sections.

24.7.1.2.1 Master/AHB-Lite

Instantiate and configure the CoreAHBLite bus as follows:

1. Select the **Memory Space** option that matches requirements:
 - If less than 16 MB of address space is required for all peripherals, select the option as shown in the following figure. This mode provides sixteen, 16 MB slots that can be used to connect up to sixteen AHB-Lite slaves.

Figure 342 • Master/AHB-Lite Memory Space Configuration – 16 MB per Slot

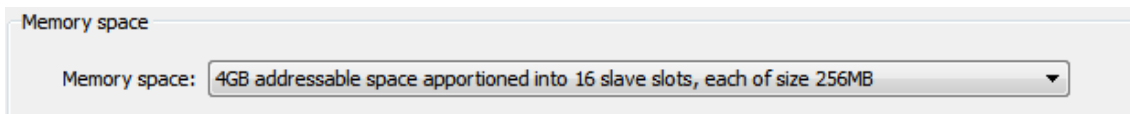


Memory space

Memory space: 256MB addressable space apportioned into 16 slave slots, each of size 16MB

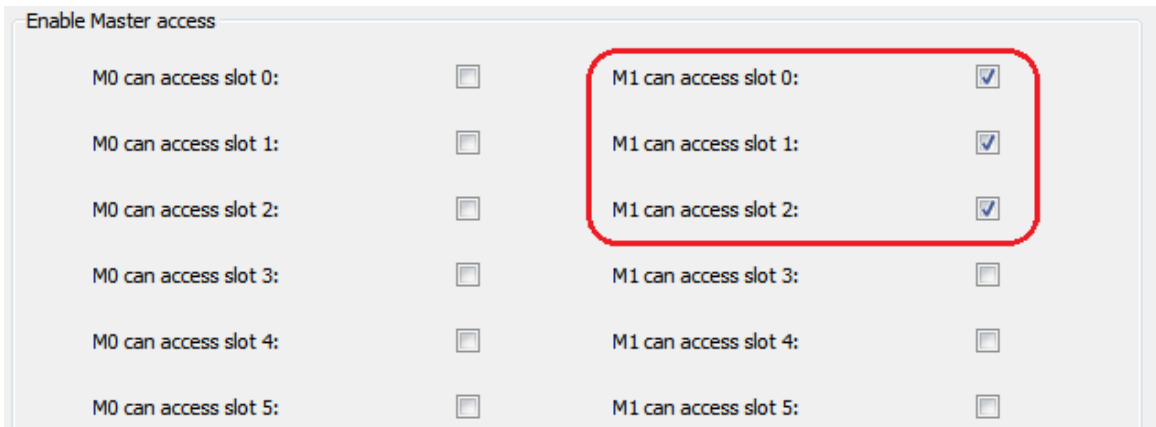
- If more than 16 MB and less than 256 MB of address space is required for any peripheral, select the option shown in the following figure. This mode provides sixteen, 256 MB slots that can be used to connect up to sixteen AHB-Lite slaves.

Figure 343 • Master/AHB-Lite Memory Space Configuration – 256 MB per Slot



2. Enable the slots planned to be used for the application. The best practice is to use the M1 to slot accesses, as shown in the following figure.

Figure 344 • Master/AHB-Lite Master Access Configuration



Notes:

- Use M1 if you plan to create a multi-master subsystem where you have a master in the fabric that requires the remap feature and thus needs to be connected to M0.
- If you have selected the 16 MB per slot option, there are no restrictions on which slots can be used.
- If you have selected the 256 MB per slot option, only the slots compatible with the FIC instance fabric memory address regions selection can be used. Each FIC memory address region is 256 MB in size. The six FIC memory regions are summarized in the following table.

Table 781 • Address Regions and Compatible Slots for 256 MB Per Slot Option

Memory Address Region	Compatible Slots
30000000-3FFFFFFF	3
50000000-5FFFFFFF	5
70000000-7FFFFFFF	7
80000000-8FFFFFFF	8
90000000-9FFFFFFF	9
F0000000-FFFFFFF	15

3. Instantiate and configure AHB-Lite compliant peripheral cores and/or custom AHB-Lite compliant components.

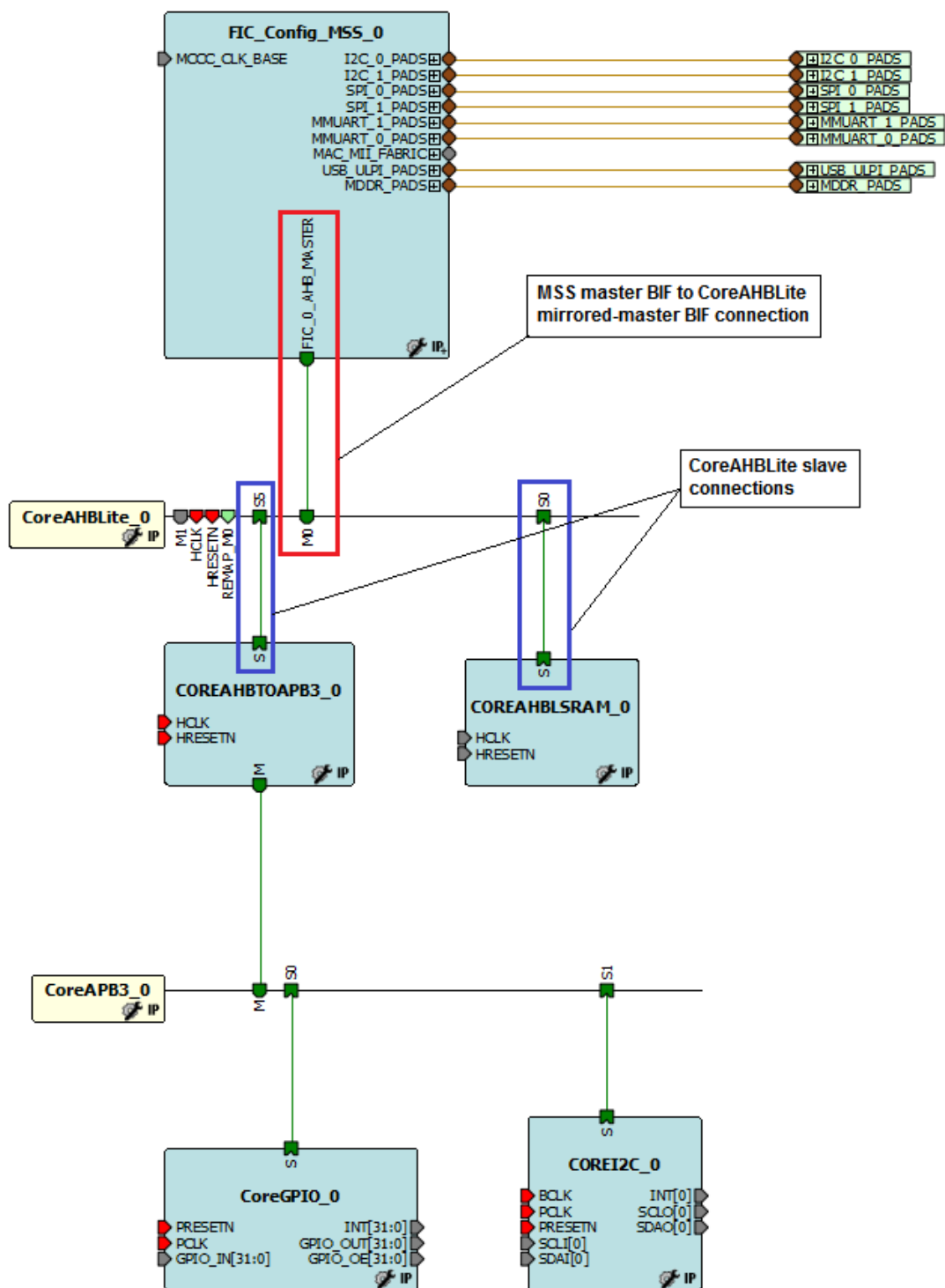
4. Connect the subsystem together; this can be done in two ways:

Automatic Connection: Right-click in the top-level SmartDesign canvas and select the Auto Connect option. This connects the FPGA fabric peripherals to the MSS FIC interfaces through the CoreAHBLite bus and CoreAPB3 bus.

Manual Connection:

- Connect the CoreAHBLite mirrored-master bus interface (BIF) port M1 to the MSS master BIF port (FIC_0/1_AHB_MASTER), as shown in the following figure.
- Connect the AHB-Lite slaves to the proper slots as per your memory map requirement.
- Clocks and resets; refer to the [Configuring the FIC Subsystem Clocks](#), page 774 and [Configuring the FIC Subsystem Reset](#), page 778.

Figure 345 • FIC Master/AHB-Lite Subsystem

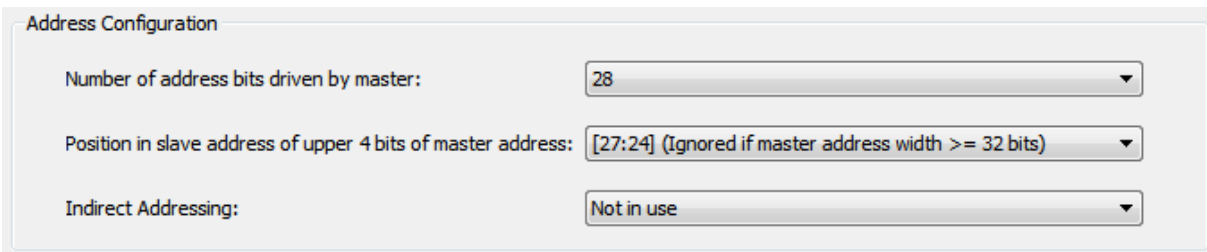


24.7.1.2.2 Master/APB

Instantiate and configure the CoreAPB3 bus as follows:

1. Select the **Address Configuration** options, as shown in the following figure. This mode provides sixteen, 16 MB slots that can be used to connect up to sixteen APB compliant slaves. If you need slots with more memory, you can combine multiple slaves to build a larger slot. Refer to the **CoreAPB3 User Guide** for more details about this option.

Figure 346 • Master/APB Address Configuration



Address Configuration

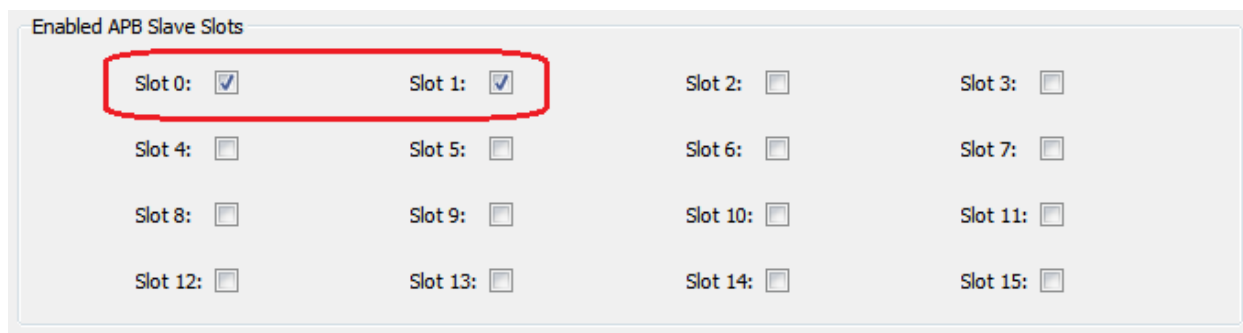
Number of address bits driven by master: 28

Position in slave address of upper 4 bits of master address: [27:24] (Ignored if master address width >= 32 bits)

Indirect Addressing: Not in use

2. Enable the slots that you are planning on using for your application, as shown in the following figure.

Figure 347 • Master/APB Slave Slots Configuration



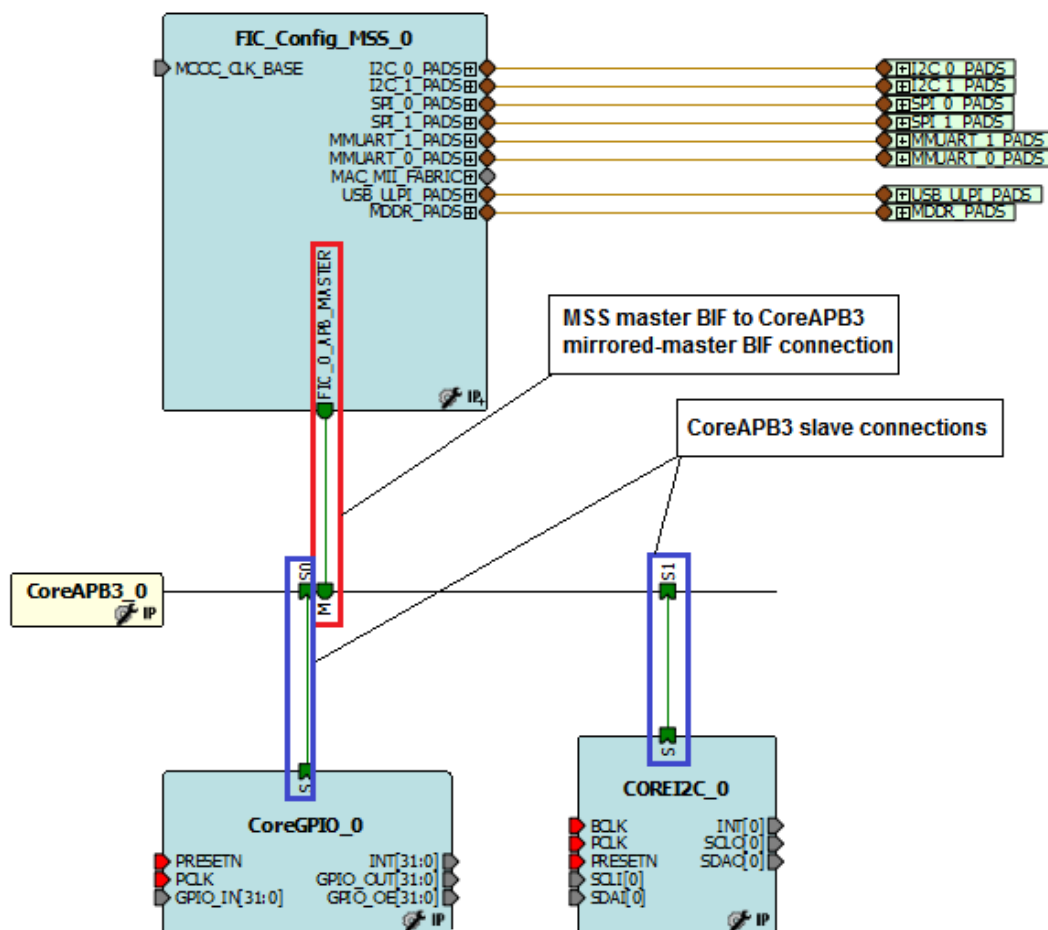
Enabled APB Slave Slots

Slot 0: <input checked="" type="checkbox"/>	Slot 1: <input checked="" type="checkbox"/>	Slot 2: <input type="checkbox"/>	Slot 3: <input type="checkbox"/>
Slot 4: <input type="checkbox"/>	Slot 5: <input type="checkbox"/>	Slot 6: <input type="checkbox"/>	Slot 7: <input type="checkbox"/>
Slot 8: <input type="checkbox"/>	Slot 9: <input type="checkbox"/>	Slot 10: <input type="checkbox"/>	Slot 11: <input type="checkbox"/>
Slot 12: <input type="checkbox"/>	Slot 13: <input type="checkbox"/>	Slot 14: <input type="checkbox"/>	Slot 15: <input type="checkbox"/>

3. Instantiate and configure APB compliant peripheral cores and/or custom APB compliant components.
4. Connect the subsystem together. this can be done in two ways.
Automatic Connection: Right-click in the top-level SmartDesign canvas and select the Auto Connect option. This connects the FPGA fabric peripherals to the MSS FIC interfaces through the CoreAPB3 bus.

Manual Connection:

- Connect the CoreAPB3 mirrored-master bus interface (BIF) port to the MSS master BIF port (FIC_0/1_APB_MASTER), as shown in the following figure.
- Connect the APB slaves to the proper slots as per your memory map requirement.
- Clocks and resets; refer to the ["Configuring the FIC Subsystem Clocks"](#) section and ["Configuring the FIC Subsystem Reset"](#) section on page 778.

Figure 348 • FIC Master/APB Subsystem

24.7.2 Configuring the FIC Subsystem Clocks

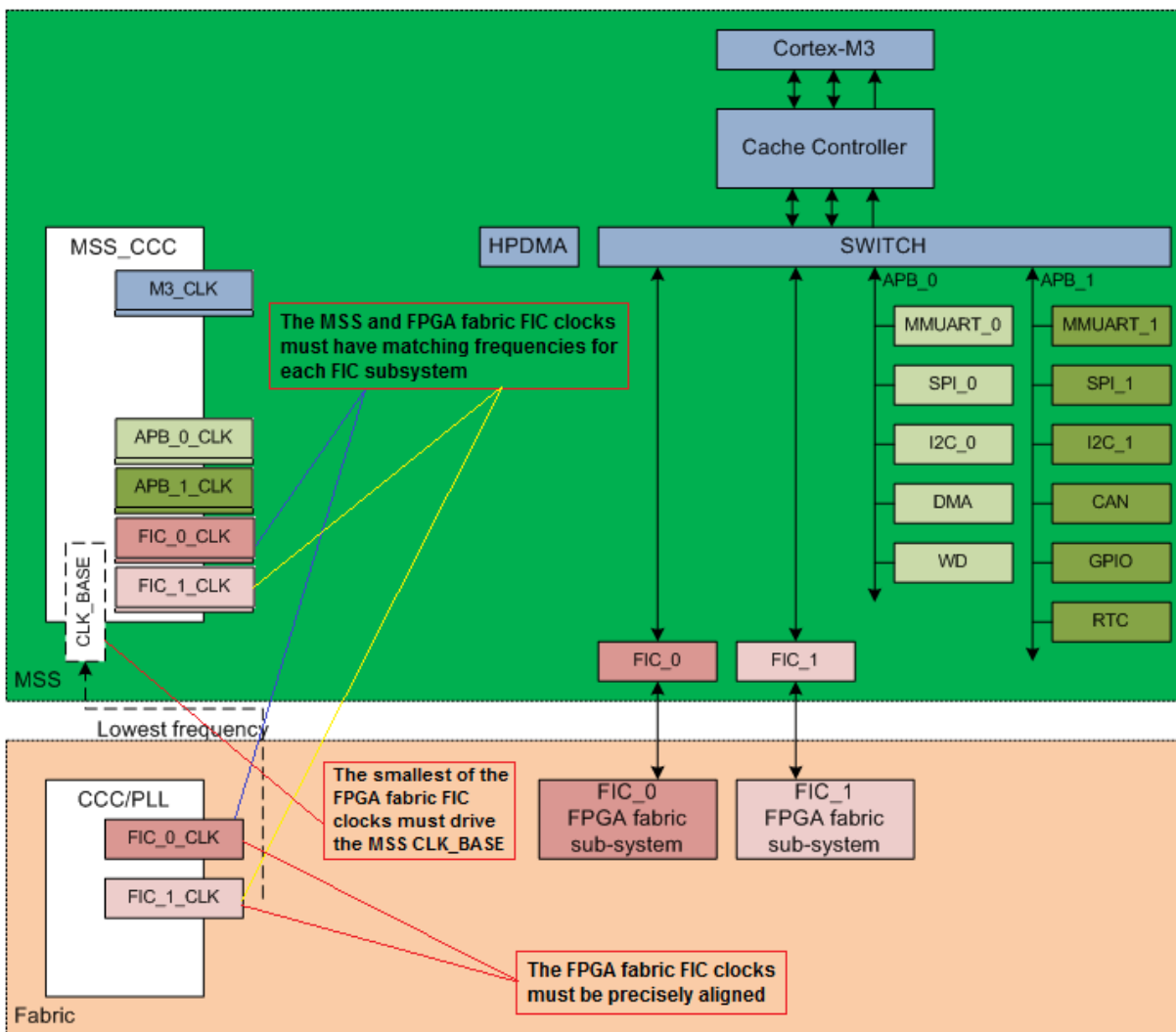
To create the proper clock configuration and connectivity you must:

- Configure the MSS CCC FIC clocks.
- Instantiate and configure an FPGA fabric CCC core.
- Connect the clock networks for each FIC subsystem.
- Connect the MSS CLK_BASE port to the correct FPGA fabric FIC subsystem clock network.

The SmartFusion2 architecture imposes the following rules that must be followed for synchronous communication between the MSS and the FPGA fabric FIC subsystems. The following figure illustrates these rules.

- Each FPGA fabric FIC subsystem must be driven by a clock whose frequency matches the frequency defined, for that particular subsystem, in the MSS_CCC configurator.
- All the FPGA fabric FIC subsystem clocks must be precisely aligned; the clocks may be of different frequencies, but the rising edges of the slower clocks must be aligned to the rising edges of the fastest clocks.
- The FPGA fabric FIC subsystem clock with the smallest frequency must drive the MSS CLK_BASE.
- If a fabric PLL is used, then the fabric PLL's LOCK output must be connected to the MSS_CCC_CLK_BASE_PLL_LOCK port, for fabric PLL lock monitoring.

Figure 349 • Clocking Scheme for Synchronous Communication Between the MSS and the FPGA Fabric

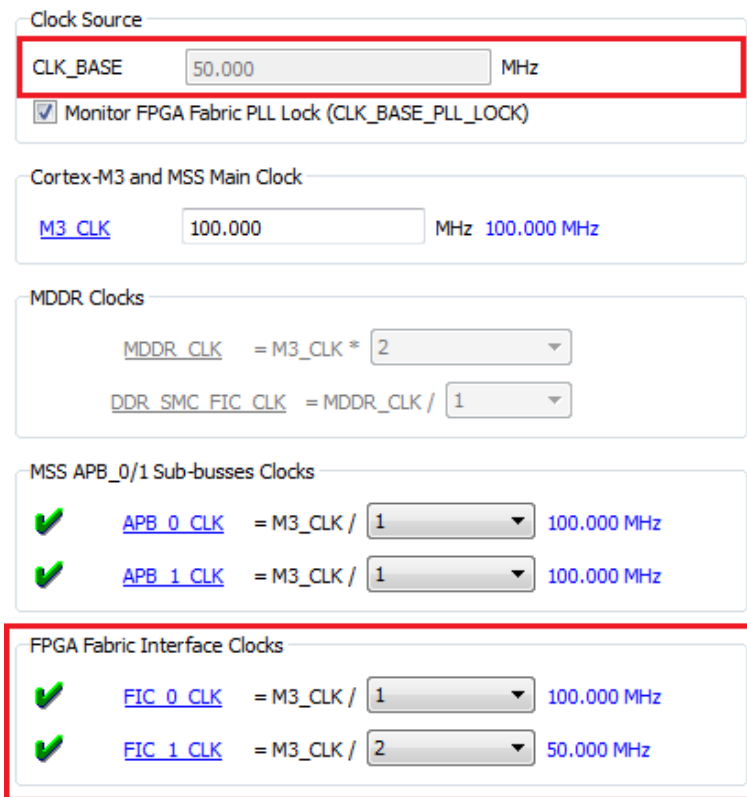


The following steps describe how to configure the clock networks for all FIC subsystems.

24.7.2.1 Step 1: Configure the MSS CCC Sub-Block

For each FIC block (FIC_0 and FIC_1) used in your design, select the clock divisors in the MSS clock configurator (MSS_CCC), as shown in the following figure.

Figure 350 • MSS CCC FIC Clock Configuration



Clock Source

CLK_BASE 50.000 MHz

☒ Monitor FPGA Fabric PLL Lock (CLK_BASE_PLL_LOCK)

Cortex-M3 and MSS Main Clock

[M3_CLK](#) 100.000 MHz 100.000 MHz

MDDR Clocks

MDDR_CLK = M3_CLK * 2

DDR_SMC_FIC_CLK = MDDR_CLK / 1

MSS APB_0/1 Sub-busses Clocks

✓ [APB_0_CLK](#) = M3_CLK / 1 100.000 MHz

✓ [APB_1_CLK](#) = M3_CLK / 1 100.000 MHz

FPGA Fabric Interface Clocks

✓ [FIC_0_CLK](#) = M3_CLK / 1 100.000 MHz

✓ [FIC_1_CLK](#) = M3_CLK / 2 50.000 MHz

Note that the CLK_BASE field is non-editable. CLK_BASE frequency, as imposed by the SmartFusion2 architecture, must be the minimum frequency of all FIC clock frequencies and is automatically computed by the MSS CCC configurator.

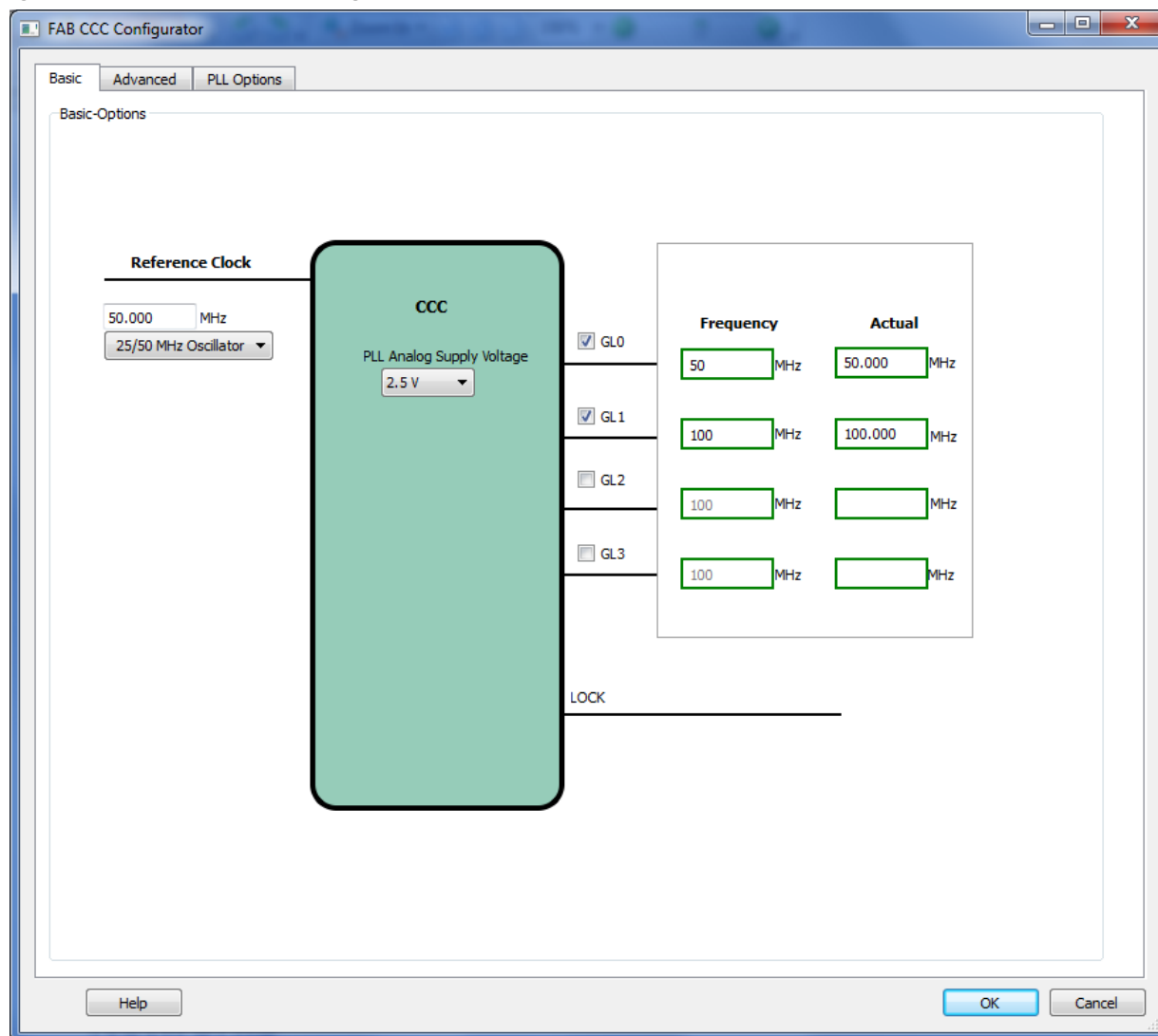
24.7.2.2 Step 2: Configure the FPGA Fabric FIC Clocks

Instantiate a CCC macro and configure it to satisfy the FIC subsystem clock rules, as described above.

Typically a global output (GLx) must be associated with each of the FIC clocks must be associated and a frequency specified for each output matching the frequencies defined in the *MSS_CCC* configurator. Microsemi recommends generating all the global outputs from a fabric PLL to guarantee the phase alignment (as shown in Figure 351, page 777).

Note: If two FIC subsystems have the same frequencies, one fabric CCC global output is sufficient for clocking both the FIC subsystems.

Figure 351 • Fabric Clocks Configuration



24.7.2.3 Step 3: Connect the FPGA Fabric FIC Subsystems Clock Networks

Connect the configured fabric CCC global outputs (GLx) to the associated FIC subsystems.

24.7.2.4 Step 4: Connect the MSS CLK_BASE Port

Connect the slowest of the fabric CCC global outputs (GLx) to MSS CLK_BASE port.

24.7.2.5 Step 5: Connect the MSS MCCC_CLK_BASE_PLL_LOCK Port

Connect the fabric CCC/PLL LOCK output to MSS MCCC_CLK_BASE_PLL_LOCK port.

24.7.2.6 Step 6: Timing Analysis Requirements

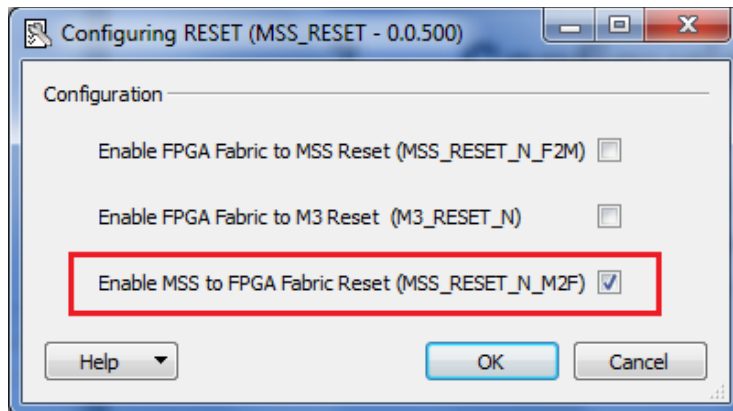
Post-layout static timing analysis must be performed to make sure that the design meets the frequency requirements defined in *MSS_CCC* and the fabric CCC configurator. M3_CLK may need to be changed or clock ratio between M3_CLK and the FIC clocks increased to get a design that passes the static timing analysis.

24.7.3 Configuring the FIC Subsystem Reset

To configure the FIC Subsystem reset:

1. Configure the MSS Reset sub-block to expose the MSS_RESET_N_M2F port, as shown in the following figure.

Figure 352 • Configure the MSS Reset Sub-Block



2. Connect the MSS_RESET_N_M2F port to all the FPGA fabric FIC subsystems reset ports.

24.7.4 Use Models

The FIC allows four possible communication scenarios that are described in the following sections.

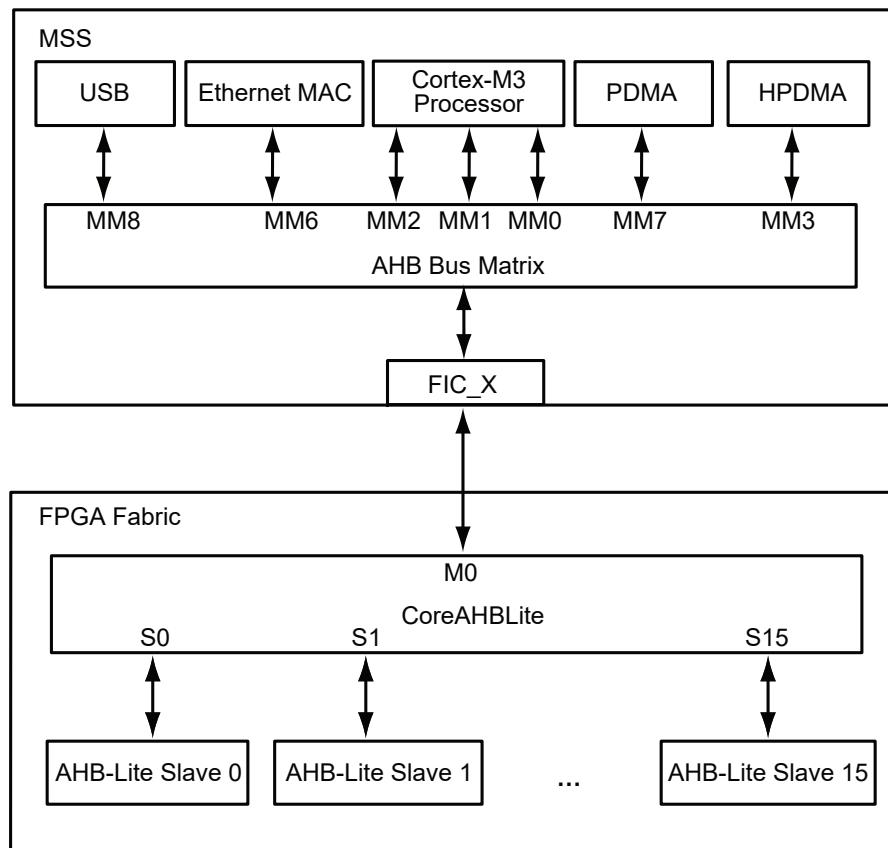
Microsemi provides numerous AHB and APB v 3.0 compliant cores in the Libero SoC IP catalog for easy instantiation into the FPGA fabric. You must instantiate CoreAHBLite and CoreAPB3 soft IP into the fabric to allow further instantiation of soft AHB-Lite and APB masters and slaves.

Note: The MSS Fabric Interface Controllers support full behavioral simulation models. Refer to [SmartFusion2 MSS BFM Simulation User Guide](#) for information.

24.7.4.1 Use Model 1: Connecting an MSS Master to the Fabric AHB-Lite Slave Interface

The following figure shows a MSS master and fabric slave scenario. The MSS acts as an AHB-Lite master for Registered or Bypass mode. The Cortex-M3 processor master, or any other master on the AHB bus matrix in the MSS, can access the AHB-Lite slaves in the fabric through FIC_0 or FIC_1. CoreAHBLite gives the HREADY and HSEL signals connectivity to the fabric AHB-Lite slaves. The MSS master AHB-Lite interface passes all incoming AHB-Lite transactions to the fabric with no error checking. If an error has occurred during the transfer, the fabric AHB-Lite slaves must signal the error condition to the master so that it is aware the transfer has been unsuccessful.

Figure 353 • AHB-Lite Slaves in the FPGA Fabric Connected to the MSS Master



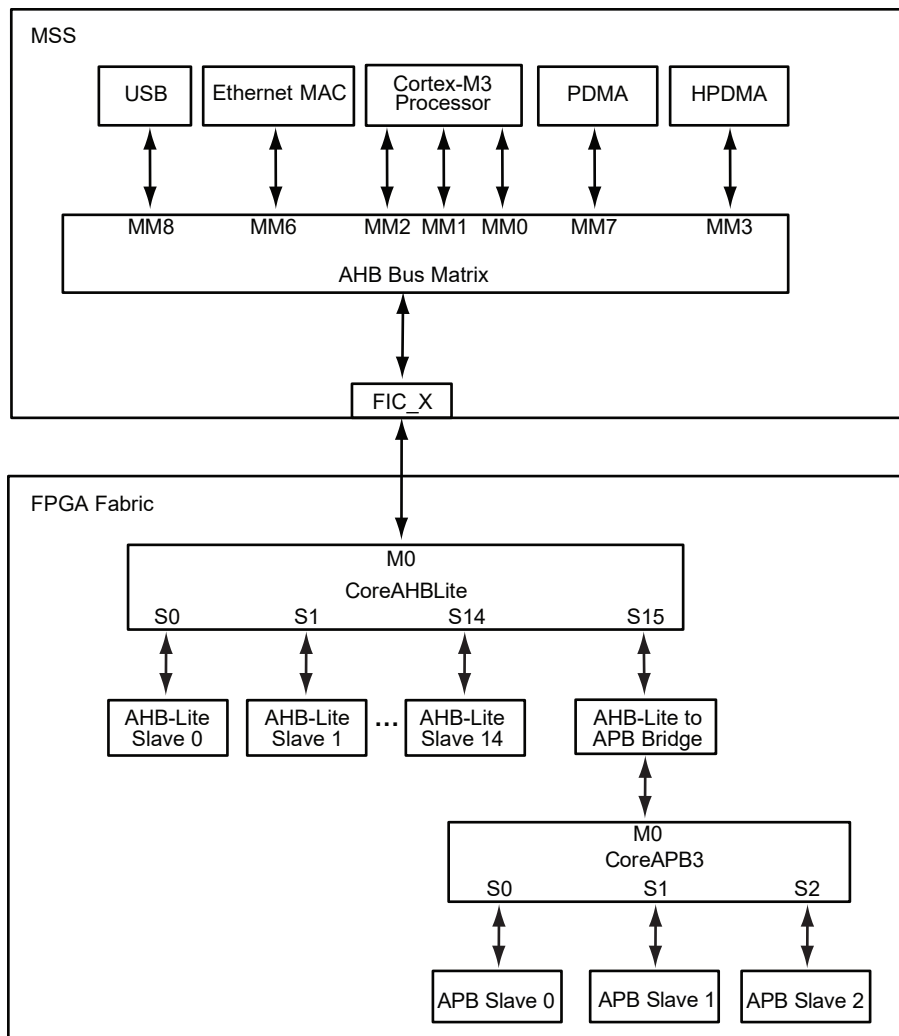
The following tutorial describes this Use Model with a design example: [TU0310: Interfacing User Logic with the Microcontroller Subsystem - Libero SoC Design Flow Tutorial](#).

This tutorial describes how to interface and handle communication between the user logic in the FPGA fabric and the MSS. The MSS is configured with FIC_0 and FIC_1 enabled. FIC_0 is configured for the AHB-Lite master interface and connected to CoreAHBLite slave. FIC_1 is configured for the APB3 master interface and connected to CoreGPIO slave.

24.7.4.2 Use Model 2: Connecting an MSS Master to the Fabric APB Slave Interface

The fabric interface allows the AHB-Lite masters in the MSS to communicate with a fabric APB v3.0 compliant slave, as shown in the following figure. A Cortex-M3 processor master, or any other master on the AHB bus matrix in the MSS, can access the APB slaves in the fabric through the fabric interface controller. CoreAHBLite gives HREADY and HSEL connectivity to the fabric AHB-Lite slaves; CoreAPB3 gives PREADY and PSEL connectivity to the fabric APB slaves.

Figure 354 • APB Slaves in the FPGA Fabric Connected to the MSS



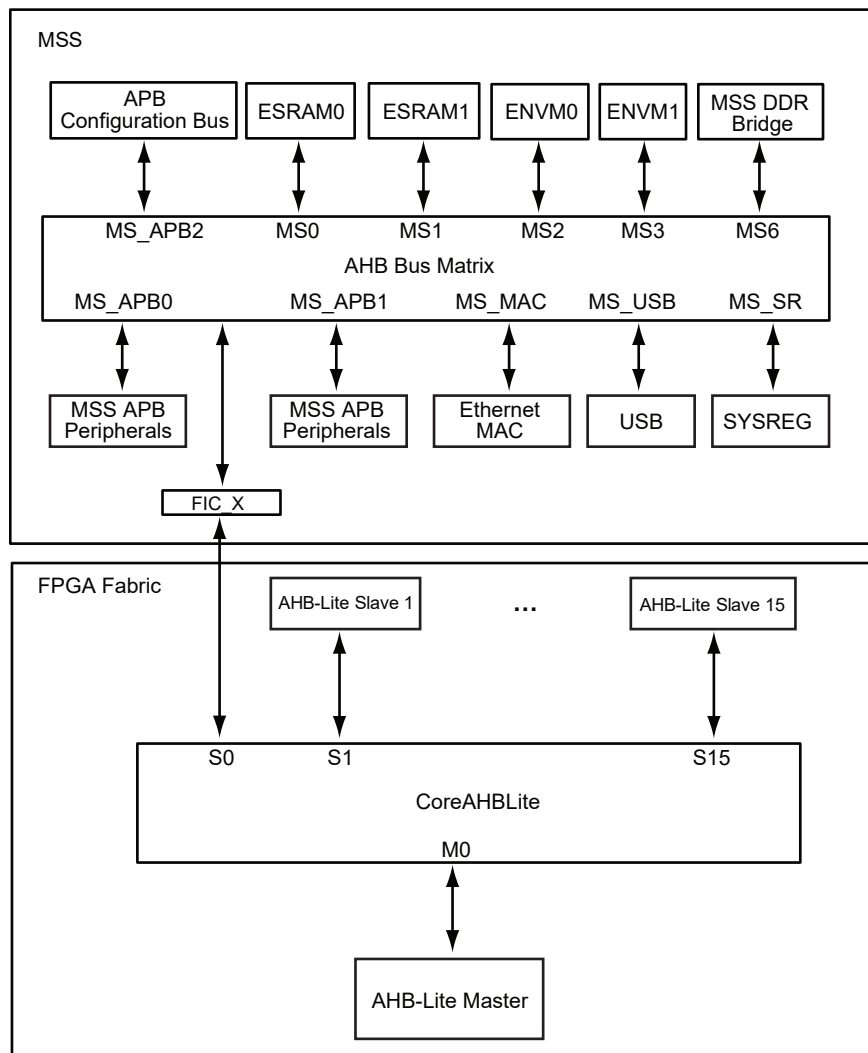
The following application note describes this Use Model with a design example: [AC392: SmartFusion2 SoC FPGA SRAM Initialization from eNVM](#).

The design example describes a method of initializing the fabric SRAM blocks after power-up with the initialization data from eNVM block using the Cortex-M3 processor as master. This design implements APB3 slave wrapper interface on the SRAM block.

24.7.4.3 Use Model 3: Connecting a Fabric AHB-Lite Master to the MSS Slave

The following figure shows the fabric AHB-Lite master connectivity with the MSS slaves and the AHB-Lite slaves implemented in the fabric. An AHB-Lite fabric master can access memory mapped peripherals in the MSS and AHB-Lite slaves in the fabric.

Figure 355 • FPGA System with the MSS Slave and the Fabric Master

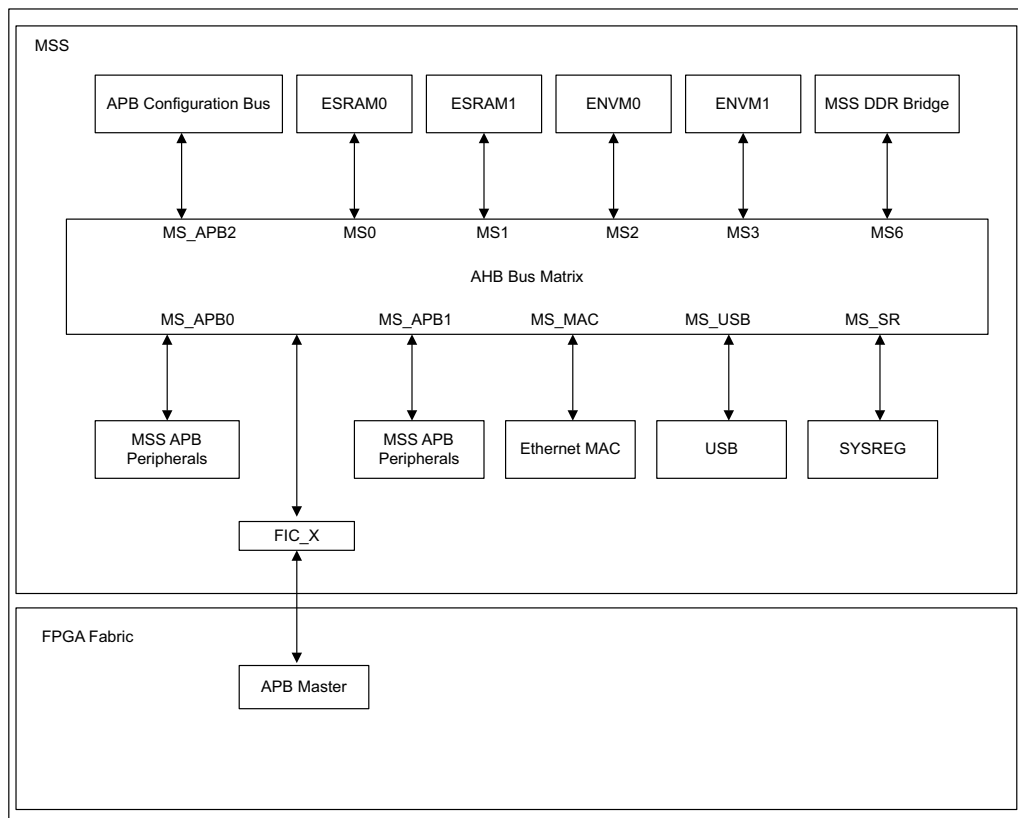


The following application note describes this Use Model with a design example: [AC388: SmartFusion2 SoC FPGA Dynamic Configuration of AHB Bus Matrix](#). The design example consists of two AHB masters in FPGA fabric that write 32-bit data to the AHB bus matrix slave eSRAM1.

24.7.4.4 Use Model 4: Connecting a Fabric APB Master to the MSS Slave

The following figure shows APB v3.0 compliant fabric master connectivity with the MSS slaves. The APB fabric master can access memory mapped peripherals in the MSS.

Figure 356 • Fabric APB Master with MSS as Slave



24.8 Reference Documents

Refer to the following documents for information on how to create an AHB-Lite or APB wrapper on the custom logic connect it to the MSS system through the FIC.

Connecting User Logic to the SmartFusion Microcontroller Subsystem Application Note: This application note explains how to create an AHB-Lite or APB wrapper on custom logic.

Building an APB3 Core for SmartFusion cSoC FPGAs Application Note: This document describes how to create an APB wrapper interface for the user logic or IP.

TU0310: Interfacing User Logic with the Microcontroller Subsystem Tutorial: This tutorial shows you how to interface and handle communication between the user logic in the FPGA fabric and the MSS. It also explains the Libero SoC design software tool flow used for designing applications for the SmartFusion2 SoC FPGA family of devices.

AMBA 3 AHB-Lite Protocol Specification:

<http://infocenter.arm.com/help/topic/com.arm.doc.ih0033a/index.html>

AMBA 3 APB Protocol Specification:

<http://infocenter.arm.com/help/topic/com.arm.doc.ih0024b/index.html>

24.9 SYSREG Control Registers for FIC_0 and FIC_1

Refer to the "**System Register Block**" section on page 670 for a detailed description of each register and bit. The following table lists the control registers for FIC_0 and FIC_1 from the SYSREG block.

Table 782 • FAB_IF Register in the SYSREG Block

Register name	Register Type	Flash Write Protect	Reset Source	Description
FAB_IF_CR	RW-P	Register	SYSRESET_N	Control register for fabric interface.
SOFT_RESET_CR	RW-P	Bit	SYSRESET_N	Generates software control interrupts to the MSS peripherals.
MSSDDR_FACC1_CR	RW-P	Field	CC_SYSRESET_N	MSS DDR fabric alignment clock controller 1 configuration register
MSSDDR_CLK_CALIB_STATUS	RO	—	SYSRESET_N	MSS DDR clock calibration status register

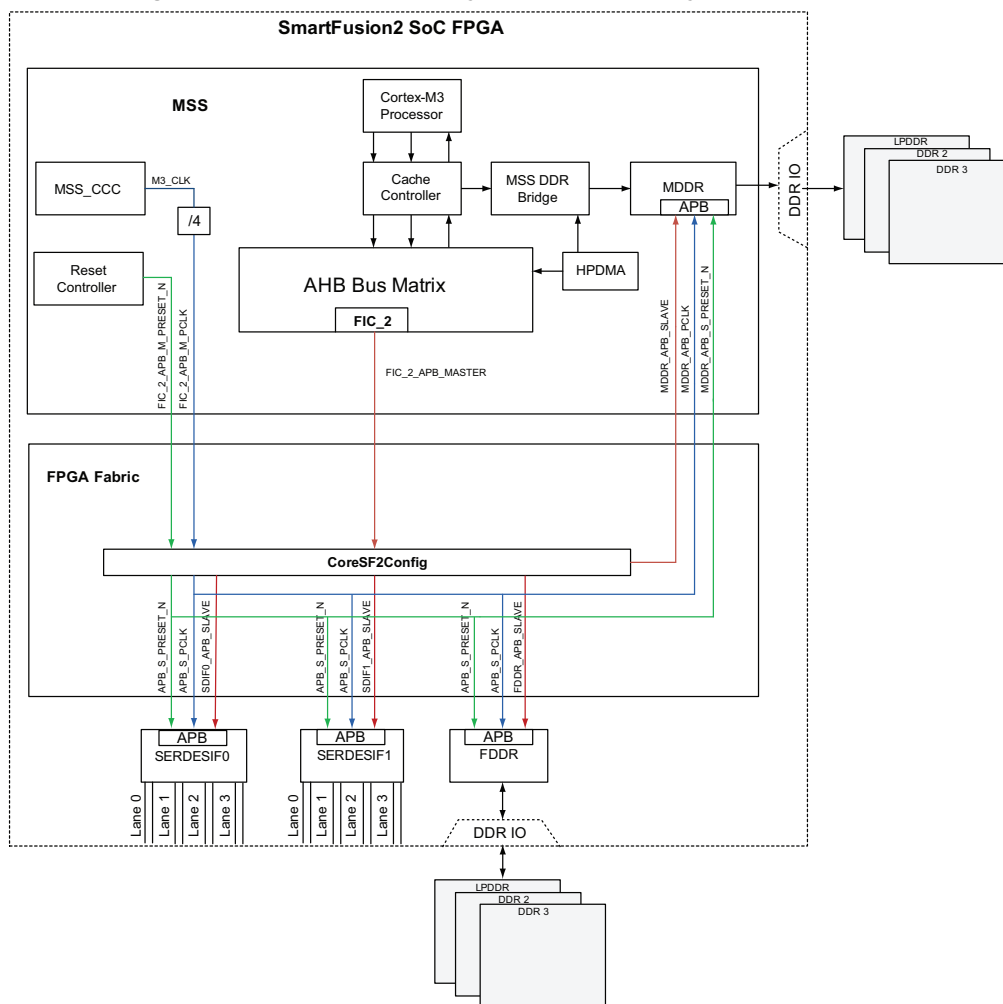
25 APB Configuration Interface

The SERDES interface (SERDESIF), fabric DDR system (FDDR), and microcontroller subsystem double data rate (MDDR) controller has to be initialized properly during bootup. Each of these subsystems contains a large number of internal registers for initialization and run-time operation. These registers are accessed through a dedicated peripheral initialization bus often called APB configuration bus. The APB configuration interface is compliant with AMBA APB3 protocol specification.

25.1 Functional Block Diagram Description

This section provides the detailed description of the FIC_2 (APB configuration bus) subsystem.

Figure 357 • APB Configuration Interface and Subsystems Connectivity with MSS Master



25.1.1 Architecture Overview

The preceding figure shows the APB configuration interfaces and SERDES and DDR subsystems connectivity with the MSS master. The AHB bus matrix FIC_2 port routes the APB configuration interface to the FPGA fabric. The SERDES and DDR subsystems are connected through CoreSF2Config soft IP. CoreSF2Config must be instantiated (available in the Libero SoC IP Catalog) in the FPGA fabric to allow configuration of FDDR, SERDESIF, and MDDR.

The following tables list the APB configuration interface signals and descriptions.

The APB configuration space is divided into multiple partitions; each partition is reserved to one specific module or type of functionality. The APB addresses are word aligned.

The base address of FDDR, SERDESIF0, and SERDESIF1 configuration address space resides at 0x40020400 and extends to address 0x4002FFFF in the memory map of the Cortex-M3 processor on the AHB bus matrix.

- Refer to the "Fabric Double Data Rate Subsystem" chapter in the [UG0446: SmartFusion2 and IGLOO2 FPGA High Speed DDR Interfaces User Guide](#) for FDDR register map details and address space partition.
- Refer to the "Serializer/Deserializer" chapter of the [UG0447: SmartFusion2 and IGLOO2 FPGA High Speed Serial Interfaces User Guide](#) for SERDES register map details and address space partition. The base address of the MDDR configuration address space resides at 0x40020000 and extends to address 0x400203FF in the memory map of the Cortex-M3 processor on the AHB bus matrix.
- Refer to the "MSS DDR Subsystem" chapter of the [UG0446: SmartFusion2 and IGLOO2 FPGA High Speed DDR Interfaces User Guide](#) for MDDR register map details and address space partition.

25.1.2 Port List

Table 783 • FDDR APB Slave Configuration Interface Port List

Port Name	Direction	Polarity	Description
APB_S_PSEL	In	High	Indicates APB slave select
APB_S_PENABLE	In	High	Indicates APB enable
APB_S_PWRITE	In	High	APB write control signal. Indicates read when Low and write when High.
APB_S_PADDR [10:2]	In		Indicates APB address. Addresses are word aligned.
APB_S_PWDATA [15:0]	In		Indicates APB write data
APB_S_PRDATA [15:0]	Out		Indicates APB read data
APB_S_PREADY	Out		Indicates APB PREADY signal and is used to extend an APB transfer.
APB_S_PSLVERR	Out	High	Indicates a transfer failure
APB_S_PCLK	In		Indicates APB clock
APB_S_PRESET_N	In	Low	Indicates APB active low reset

Table 784 • MDDR APB Slave Configuration Interface Port List

Port Name	Direction	Polarity	Description
MDDR_APB_S_PSEL	In	High	Indicates APB slave select
MDDR_APB_S_PENABLE	In	High	Indicates APB enable
MDDR_APB_S_PWRITE	In	High	APB write control signal. Indicates read when Low and write when High.
MDDR_APB_S_PADDR [10:2]	In		Indicates APB address. Addresses are word aligned.
MDDR_APB_S_PWDATA [15:0]	In		Indicates APB write data
MDDR_APB_S_PRDATA [15:0]	Out		Indicates APB read data
MDDR_APB_S_PREADY	Out		Indicates APB PREADY signal and used to extend an APB transfer.
MDDR_APB_S_PSLVERR	Out	High	Indicates a transfer failure
MDDR_APB_S_PCLK	In		Indicates APB clock
MDDR_APB_S_PRESET_N	In	Low	Indicates APB active low reset

Table 785 • SERDESIF APB Slave Configuration Interface Port List

Port Name	Direction	Polarity	Description
APB_S_PSEL	In	High	Indicates APB slave select
APB_S_PENABLE	In	High	Indicates APB enable
APB_S_PWRITE	In	High	Indicates APB write control signal. Indicates read when Low and write when High.
APB_S_PADDR [13:2]	In		Indicates APB address. Addresses are word aligned.
APB_S_PWDATA [31:0]	In		Indicates APB write data
APB_S_PRDATA [31:0]	Out		Indicates APB read data
APB_S_PREADY	Out		Indicates APB PREADY signal and used to extend an APB transfer.
APB_S_PSLVERR	Out	High	Indicates a transfer failure
APB_S_PCLK	In		Indicates APB clock
APB_S_PRESET_N	In	Low	Indicates APB active low reset

Table 786 • MSS APB Master Configuration Interface Port List

Port Name	Direction	Polarity	Description
FIC_2_APB_M_PSEL	Out	High	Indicates APB slave select
FIC_2_APB_M_PENABLE	Out	High	Indicates APB enable
FIC_2_APB_M_PWRITE	Out	High	APB write control signal. Indicates read when Low and write when High.
FIC_2_APB_M_PADDR [15:2]	Out		Indicates APB address. Addresses are word aligned.
FIC_2_APB_M_PWDATA [31:0]	Out		Indicates APB write data
FIC_2_APB_M_PRDATA [31:0]	In		Indicates APB read data
FIC_2_APB_M_PREADY	In		Indicates APB PREADY signal and used to extend an APB transfer.
FIC_2_APB_M_PSLVERR	In	High	Indicates a transfer failure
FIC_2_APB_M_PCLK	In		Indicates APB clock
FIC_2_APB_M_PRESET_N	In	Low	Indicates APB active low reset

25.1.3 CoreSF2Config Soft IP

CoreSF2Config facilitates configuration of peripheral blocks (MDDR, FDDR, and SERDESIF blocks) in a SmartFusion2 device, as shown in [Figure 358](#), page 787. CoreSF2Config has a mirrored master APB port and several mirrored slave APB ports. The mirrored master port should be connected to the FIC_2_APB_MASTER port of the MSS and the mirrored slave ports should be connected to the APB slave ports of the blocks to be configured.

CoreSF2Config soft IP is available in the Libero SoC IP Catalog. Refer to the **CoreSF2Config Handbook** for port lists and their descriptions, design flows, memory maps, and Control and Status register details.

25.2 How to Use

This section describes how to use the FIC_2 (Peripheral Initialization) subsystem in the design.

25.2.1 Configuring FIC_2 (Peripheral Initialization) Using Libero SoC

This section describes the FIC_2 (Peripheral Initialization) configuration in Libero SoC and shows options available for configuring FIC_2. The FIC_2 is not configured by default in the MSS configurator when the Libero SoC project is created. The following steps are required.

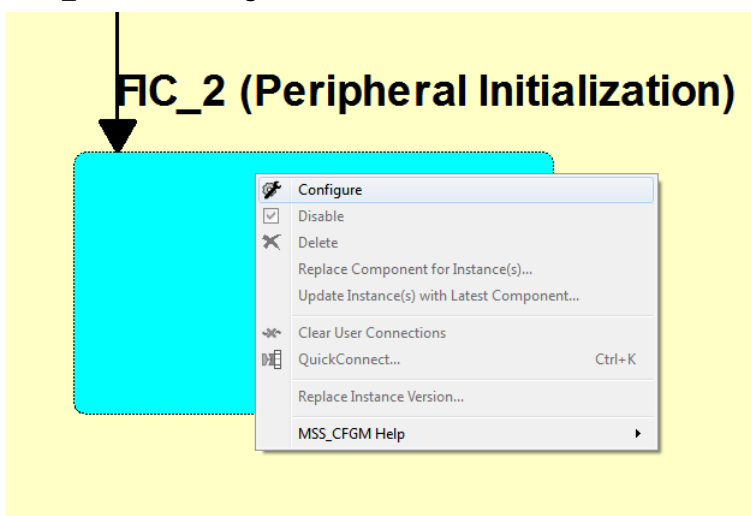
25.2.1.1 Step 1

Instantiate the SmartFusion2 MSS component into the Libero project and configure (enable/disable) the peripherals using MSS configurator, as required.

25.2.1.2 Step 2

Double-click **FIC_2 (Peripheral Initialization)** or right-click **FIC_2** and select **Configure**, as shown in the following figure.

Figure 358 • Configure FIC_2 in MSS Configurator

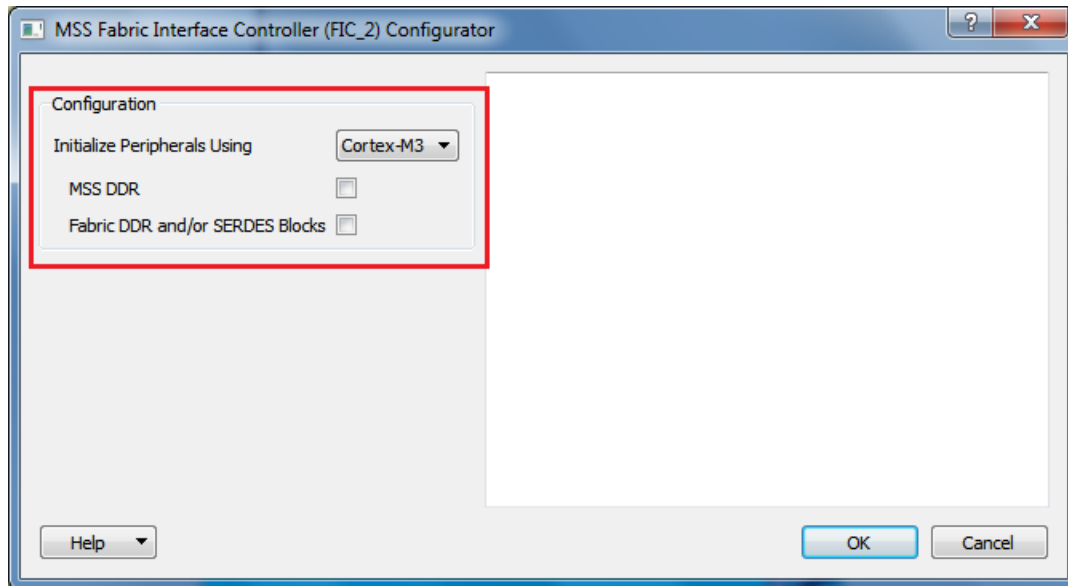


The following options are available in the APB configuration interfaces of SERDES and DDR subsystems.

- Initialize peripherals using Cortex-M3 processor
 - MSS DDR
 - Fabric DDR and/or SERDES Blocks

The following figure shows the FIC_2 configurator.

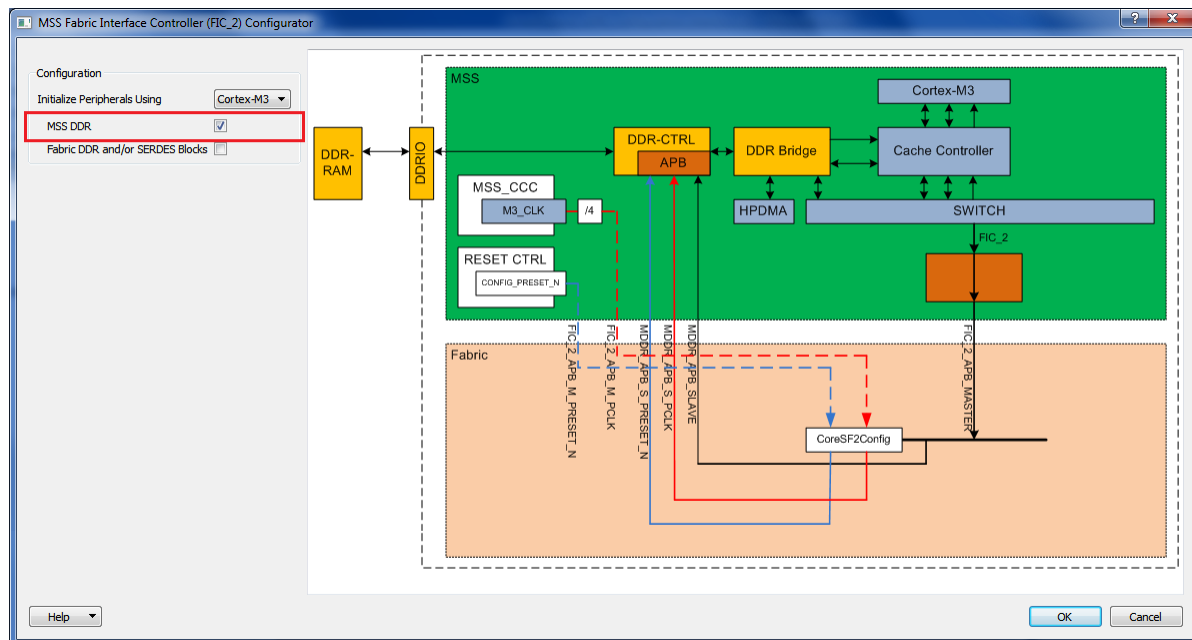
Figure 359 • FIC_2 Configurator



25.2.1.3 Step 3

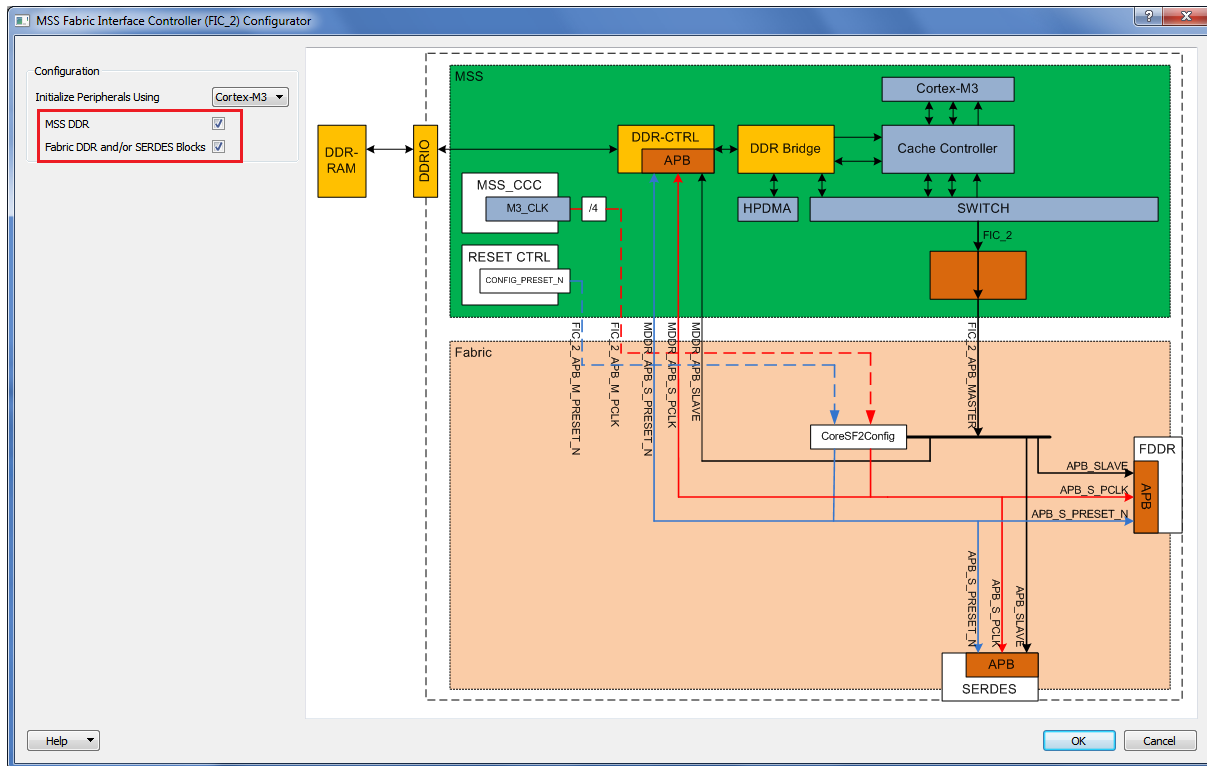
Select either or both check boxes as required. If MSS DDR is selected, the FIC_2 Configurator shows the graphical illustration of the connectivity between the FIC_2 APB master and MSS DDR APB slave through CoreSF2Config, as shown in the following figure.

Figure 360 • FIC_2 Configuration for MSS DDR



CoreSF2Config must be instantiated in SmartDesign.connections made as illustrated in FIC_2 configurator. The following figure shows the connectivity between APB configuration interfaces of the SERDES and DDR subsystems (both are selected).

Figure 361 • FIC_2 Configuration for MSS DDR, FDDR, and SERDES



If System Builder is used for creating the design, CoreSF2Config is Instantiated and connections are made automatically.

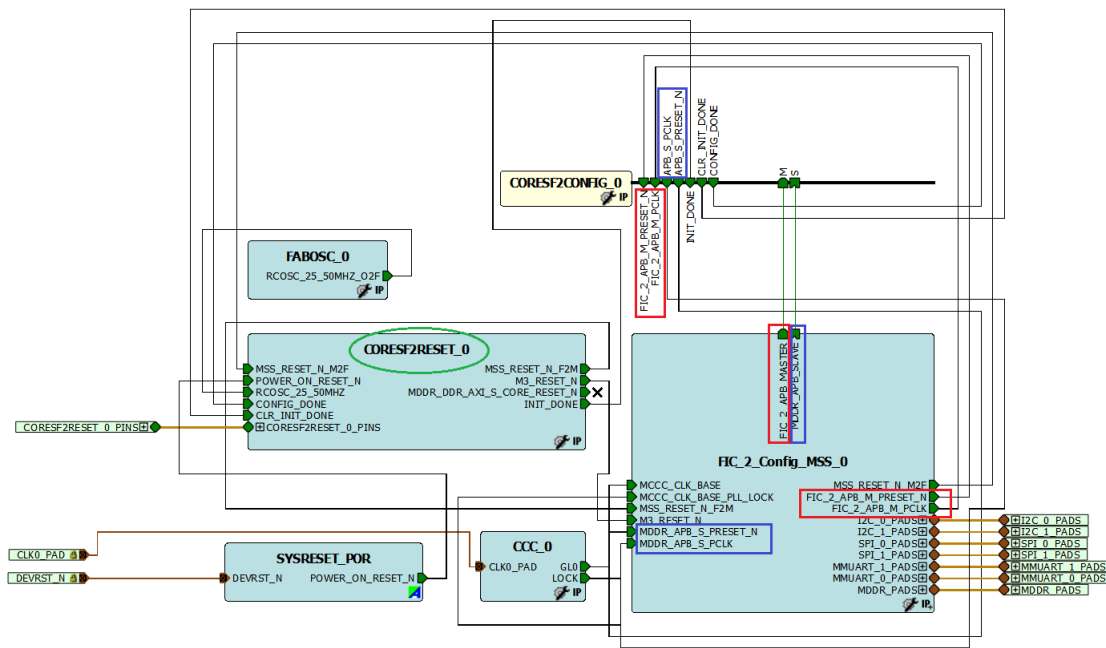
25.2.2 FIC_2 Use Models

This section explains the use models and provides directions for using FIC_2 in an application. The design is created using the System Builder flow.

25.2.2.1 Use Model 1: Configuring MSS DDR

- Select **Use System Builder** while creating a new project from the Design Templates and Creators panel in Libero SoC.
- Follow the steps in the **System builder - Device Features** GUI with default settings and generate the design. The following figure shows the generated design when opened in SmartDesign.

Figure 362 • MSS DDR Design with APB Configuration Interface

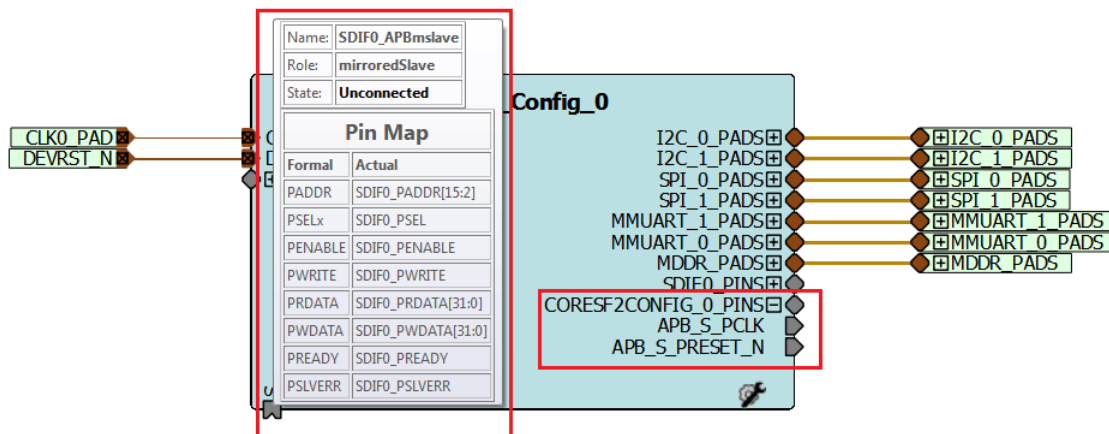


FIC_2 APB master signals are shown in the boxes outlined in red, while MSS DDR APB slave signal are shown in the boxes outlined in blue. These signals are connected to CoreSF2Config. In addition to APB configuration signals, CoreSF2Config has a few other signals (CONFIG_DONE, CLR_INIT_DONE, and INIT_DONE) that are connected to CoreSF2Reset. CoreSF2Reset handles sequencing of reset signals in SmartFusion2 devices. It is particularly concerned with resets related to peripheral blocks (MDDR, FDDR, and SERDESIF blocks). CoreSF2Reset soft IP is available in the Libero SoC IP catalog. Refer to the **CoreSF2Reset Handbook** for port lists, port descriptions, and design flow.

25.2.2.2 Use Model 2: Configuring SERDES

- Select **Use System Builder** while creating a new project from the Design Templates and Creators panel in Libero SoC.
- Select **SERDESIF_0** in the **System builder - Device Features** GUI. Follow the rest of the steps with default settings and generate the design. The following figure shows the top-level components of the generated design.

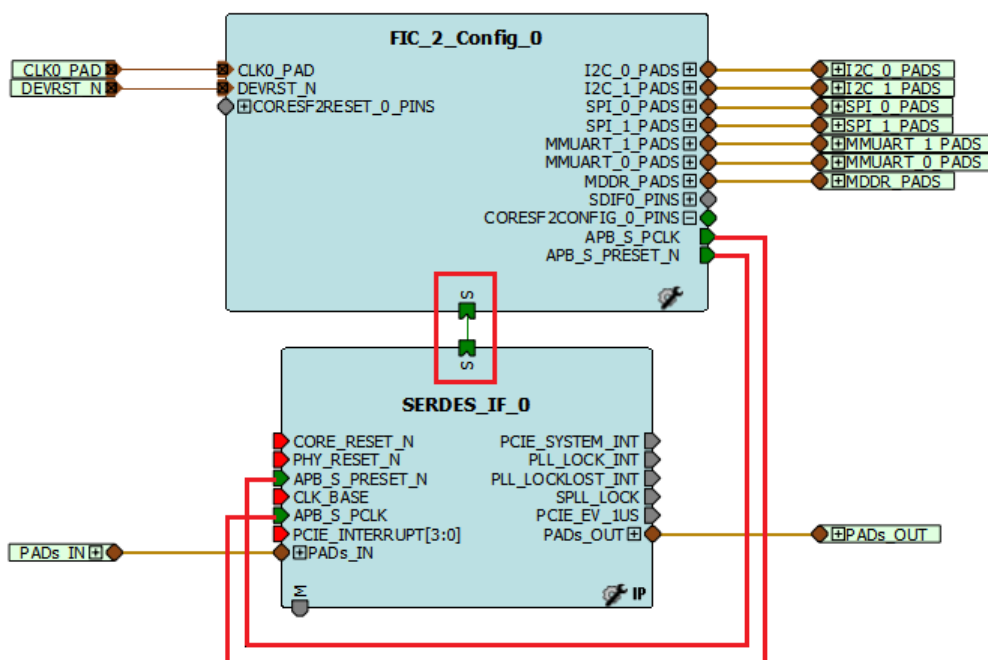
Figure 363 • Top-Level Components with APB Configuration Interface Signals



The signals in the boxes outlined in red show the CoreSF2Config mirrored APB slave port that should be connected to the APB slave port of the SERDES block to be configured. If you open this component in SmartDesign, the canvas shows a hierarchical view of the design components (for example, CoreSF2Config and CoreSF2Reset) with APB configuration interfaces, as shown in [Figure 362](#), page 790.

- Instantiate the High Speed Serial Interface (SERDES_IF) macro in SmartDesign and connect the CoreSF2Config mirrored APB slave port with APB slave port of the SERDES_IF block, as shown in the following figure.

Figure 364 • Interfacing of CoreSF2Config Mirrored APB Slave with SERDES_IF Block



26 Error Detection and Correction Controllers

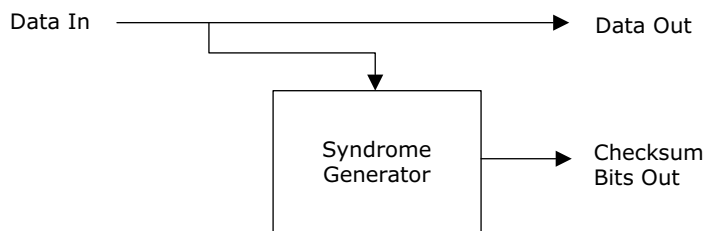
In SEU susceptible environments, which are increasingly more common at ground-level due to shrinking geometries, storage elements such as RAMs and FIFOs in the microcontroller subsystem (MSS) are susceptible to transient errors caused by heavy ions. Errors can be detected and corrected by employing error detection and correction (EDAC). The EDAC controllers implemented in SmartFusion2 devices support single error correction and double error detection (SEDED).

SmartFusion2 SOC memories such as eSRAM, cache, eNVM, USB internal memory, internal FIFOs of the Ethernet MAC, and the internal RAM of the controller area network (CAN) controller are protected by EDAC. Fabric SRAM blocks are not protected by EDAC.

26.1 Functional Description

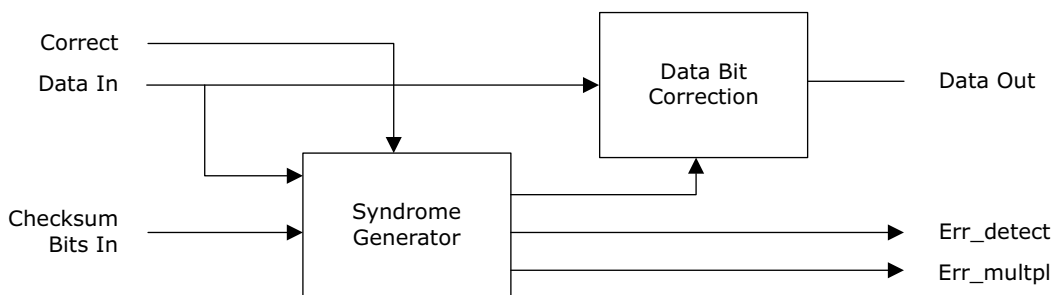
EDAC algorithms generate additional checksum bits for each data word to be stored. The checksum bits are generated and written to the memory when the data bits are written, as shown in the following figure. When the word is read, the checksum bits are utilized to determine whether one or more bits are in error. If the error is a single-bit error, checksum bits determine which bit contains the error. EDAC algorithms implemented in SmartFusion2 SOC devices are designed to detect all two-bit errors and correct all single-bit errors within a single word.

Figure 365 • EDAC in Write Mode



When data is written to a storage element in memory through EDAC, checksum bits are generated based on the input data pattern. The checksum bits, along with the input data, are stored in the target memory. This permits the data to propagate to the data storage element unchanged while the appropriate checksum bits are generated to be stored along with the data.

Figure 366 • EDAC in Read Mode (Reading From Memory)



For a read operation, the checksum bits are computed from the Data In inputs, and XORed with the Checksum Bits In inputs to form the error syndrome. The error syndrome is internally generated and is not user-accessible.

If all bits of the error syndrome are 0, then there is no error. If one or more syndrome bits are 1, then an error is detected as indicated by the output Err_detect = 1. If the error is not correctable (2 or more bits errored), then it is indicated by the output Err_multpl = 1. The error syndrome indicates which bit is in

error, or whether multiple bits are in error, and is useful in systems where it is desirable to keep a log of which bits have been in error.

When a correctable error is detected (indicated by Err_detect = 1 and Err_multpl = 0), the data bit in error is corrected as it passes from Data In to Data Out. If the error is not correctable (indicated by Err_multpl = 1), data passes unchanged from Data In to Data Out.

There are interrupts associated with errors. These are covered in the chapter for each memory type.

26.1.1 EDAC Checksum Bits Width

The following table lists the data width ranges and the corresponding checksum bit widths.

Table 787 • Minimum Number of Checksum Bits Required Data

Width	Checksum Bits
27 to 57	7
58 to 64	8

26.2 Configuration

The EDAC architecture is implemented to protect different types of memories. The data and checksum bit widths of the EDAC change according to the memory specifications.

Refer to the following chapters for EDAC configurations for specific types of memories:

- eSRAM: Embedded SRAM (eSRAM) Controllers
- Internal FIFOs of the Ethernet MAC: Ethernet MAC
- USB internal memory: Universal Serial Bus OTG Controller
- Internal RAM of the CAN controller: CAN Controller
- eNVM: Embedded Nonvolatile Memory (eNVM) Controllers

In the [UG0446: SmartFusion2 and IGLOO2 FPGA High Speed DDR Interfaces User Guide](#), refer to the following chapters:

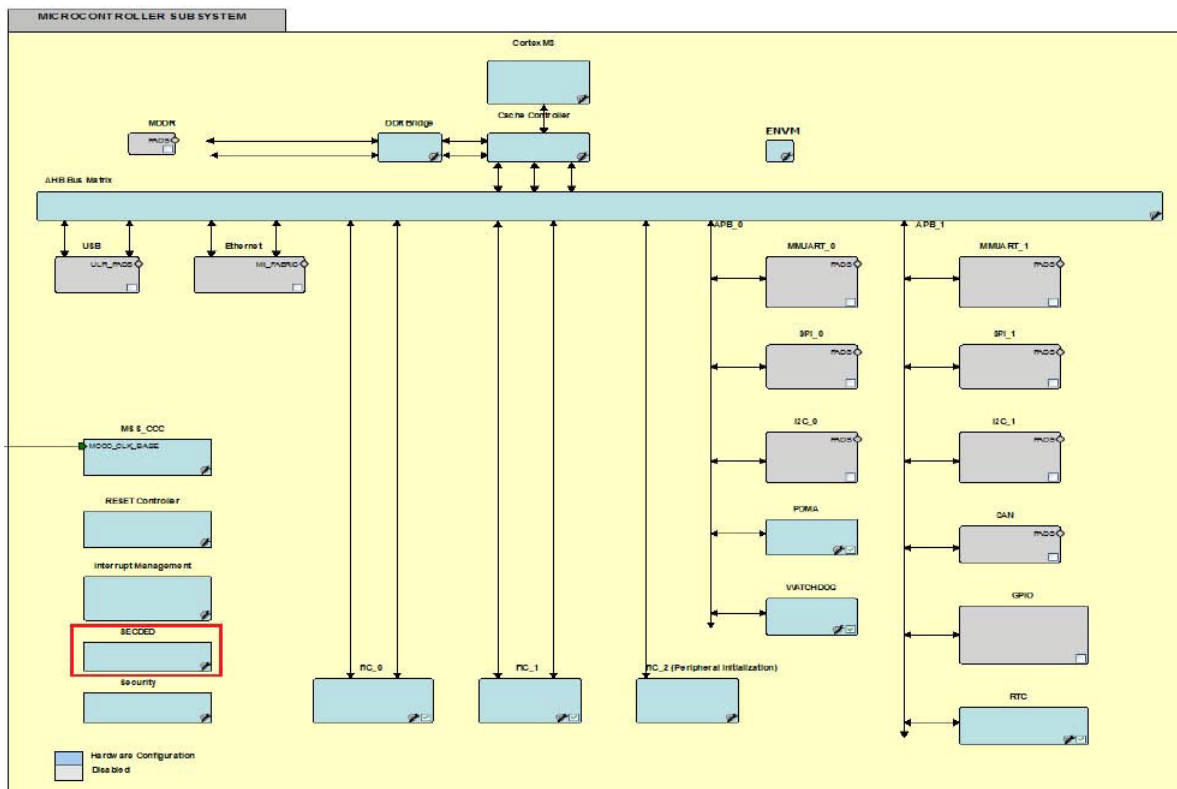
- MDDR: MDDR Subsystem
- FDDR: Fabric DDR Subsystem

26.3 How to Use EDAC

EDAC can be configured using the SECDED configurator available in the SmartFusion2 SOC, as shown in the following figure. Using the SECDED configurator, EDAC options for the following memories can be configured:

- eSRAM (eSRAM0, eSRAM1)
- Cache
- Ethernet MAC transmit and receive memory
- USB internal memory
- CAN internal memory
- MDDR

Figure 367 • EDAC in Read Mode (Reading From Memory)

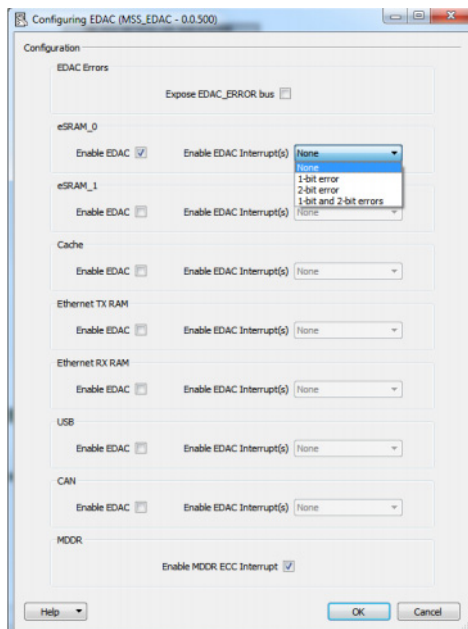


The following figure shows the SECDED configurator options GUI available in the MSS. The SECDED configurator dialog box is organized as follows:

- **EDAC_ERROR BUS:** This is the EDAC_ERROR bus signal to the FPGA fabric. This signal can be used to expose the error bus to the fabric for monitoring.
- **EDAC_ENABLE:** EDAC_ENABLE can be used to enable the EDAC functionality for each of the following blocks: eSRAM0, eSRAM1, Cache, Ethernet MAC Tx and Rx RAMS, USB, CAN, and MDDR.
- **Enable EDAC Interrupt(s):** Enable EDAC Interrupt(s) can be used to enable the interrupts for each of the following blocks: eSRAM0, eSRAM1, Cache, Ethernet MAC TX and RX RAMs, USB, and CAN. Selection options for enable interrupts are available as below:
 - None
 - 1-bit error
 - 2-bit error
 - 1-bit and 2-bit errors

- **Enable MDDR ECC Interrupt:** Enable MDDR ECC Interrupt can be used to enable the MSS DDR (MDDR) ECC interrupts.

Figure 368 • EDAC in Read Mode (Reading From Memory)



The values entered in the configurator will be exported into the programming files for programming the flash bits that control the EDAC functionality. The flash bits are loaded in the system registers at power-up (or when the DEVRST_N external pad is asserted/deasserted).